```
        Ursina cheat sheet

        This document lists most modules and classes in ursina. Each
section is structured as follows:

        ClassName(BaseClass)
            module location

            parameters
                How instantiate the class, ie. Button(text='',
**kwargs).
                '**kwargs' in this case, means you can give it optional
keyword arguments.
                For example, Button('Start', scale=.25,
color=color.blue, position=(-.1,.25)) also includes
                information on how big the button should be, its color
and its position.

            attributes
                Names of values we can get/set, sometimes followed by
its starting value and a short explanation.
                For example, 'scale', 'color' and 'position' are
                attributes we gave the Button above. These are members
of Entity, which Button class
                inherits from, so the Button class can also access
these.

            methods/functions
                these ends with (), which means they are functions that
can be called.
                Also lists their parameters and default arguments.
                For example, Entity has a method called 'look_at()'. You
need to give it a
                'target' (an Entity or position) to look at and
optionally say
                which axis will be facing the target.

            example

        You can search the document with Ctrl+F for instant search
results.
```

Entity()

ursina.entity

Entity(add_to_scene_entities=True, **kwargs)

```
name = camel_to_snake(self.type)
enabled = True     # disabled entities will not be visible nor run
code.
visible = True
ignore = False     # if True, will not try to run code.
eternal = False    # eternal entities does not get destroyed on
scene.clear()
ignore_paused = False      # if True, will still run when application
is paused. useful when making a pause menu for example.
ignore_input = False
parent = scene     # default parent is scene, which means it's in 3d
space. to use UI space, set the parent to camera.ui instead.
add_to_scene_entities = add_to_scene_entities # set to False to be
ignored by the engine, but still get rendered.
model = None       # set model with model='model_name' (without file
type extension)
color = color.white
texture = None     # set model with texture='texture_name'. requires a
model to be set beforehand.
render_queue = 0
double_sided = False
collision = False  # toggle collision without changing collider.
collider = None    # set to 'box'/'sphere'/'mesh' for auto fitted
collider.
scripts = list()   # add with add_script(class_instance). will assign
an 'entity' variable to the script.
animations = list()
hovered = False    # will return True if mouse hovers entity.
origin = Vec3(0,0,0)
```

```
position = Vec3(0,0,0) # right, up, forward. can also set self.x,
self.y, self.z
rotation = Vec3(0,0,0) # can also set self.rotation_x,
self.rotation_y, self.rotation_z
scale = Vec3(1,1,1)    # can also set self.scale_x, self.scale_y,
self.scale_z
line_definition = None # returns a Traceback(filename, lineno,
function, code_context, index).
world_parent
type                # get class name.
types               # get all class names including those this
inhertits from.
visible_self        # set visibility of self, without affecting
children.
origin_x
origin_y
origin_z
world_position
world_x
world_y
world_z
x
y
z
X                   # shortcut for int(entity.x)
Y                   # shortcut for int(entity.y)
Z                   # shortcut for int(entity.z)
world_rotation
world_rotation_x
world_rotation_y
world_rotation_z
rotation_x
rotation_y
rotation_z
quat
world_scale
world_scale_x
world_scale_y
world_scale_z
scale_x
scale_y
scale_z
transform           # get/set position, rotation and scale
world_transform     # get/set world_position, world_rotation and
world_scale
forward             # get forward direction.
back                # get backwards direction.
right               # get right direction.
left                # get left direction.
up                  # get up direction.
down                # get down direction.
screen_position     # get screen position(ui space) from world space.
shader
texture_scale
texture_offset
tileset_size        # if the texture is a tileset, say how many tiles
there are so it only use one tile of the texture, e.g. tileset_size=
[8,4]
tile_coordinate     # set the tile coordinate, starts in the lower
left.
alpha
always_on_top
unlit
billboard           # set to True to make this Entity always face the
camera.
model_bounds
model_center
bounds
children
attributes          # attribute names. used by duplicate().

enable()
disable()
set_shader_input(name, value)
generate_sphere_map(size=512,
name=f'sphere_map_{len(scene.entities)}')
generate_cube_map(size=512, name=f'cube_map_{len(scene.entities)}')
reparent_to(entity)
get_position(relative_to=scene)
set_position(value, relative_to=scene)
```

```
add_script(class_instance)
combine(analyze=False, auto_destroy=True, ignore=[])
flip_faces()
look_at(target, axis='forward')
look_at_2d(target, axis='z')
has_ancestor(possible_ancestor)
animate(name, value, duration=.1, delay=0, curve=curve.in_expo,
loop=False, resolution=None, interrupt='kill', time_step=None,
auto_destroy=True)
animate_position(value, duration=.1, **kwargs)
animate_rotation(value, duration=.1,  **kwargs)
animate_scale(value, duration=.1, **kwargs)
animate_{e}(value, duration=.1, delay=0, **kwargs)
shake(duration=.2, magnitude=1, speed=.05, direction=(1,1))
animate_color(value, duration=.1, interrupt='finish', **kwargs)
fade_out(value=0, duration=.5, **kwargs)
fade_in(value=1, duration=.5, **kwargs)
blink(value=color.clear, duration=.1, delay=0,
curve=curve.in_expo_boomerang, interrupt='finish', **kwargs)
intersects(traverse_target=scene, ignore=(), debug=False)
```

```python
e = Entity(model='quad', color=color.orange, position=(0,0,1),
scale=1.5, rotation=(0,0,45), texture='brick')

'''example of inheriting Entity'''
class Player(Entity):
    def __init__(self, **kwargs):
        super().__init__()
        self.model='cube'
        self.color = color.red
        self.scale_y = 2

        for key, value in kwargs.items():
            setattr(self, key, value)

    def input(self, key):
        if key == 'space':
            self.animate_x(2, duration=1)

    def update(self):
        self.x += held_keys['d'] * time.dt * 10
        self.x -= held_keys['a'] * time.dt * 10

player = Player(x=-1)
```

Text(Entity)

ursina.text

Text(text='', start_tag=start_tag, end_tag=end_tag, ignore=True, **kwargs)

```
size = Text.size
parent = camera.ui
text_nodes = list()
images = list()
origin = (-.5, .5)
font = Text.default_font
resolution = Text.default_resolution
line_height = 1
use_tags = True
start_tag = start_tag
end_tag = end_tag
text_colors = {'default' : color.text_color}
tag = Text.start_tag+'default'+Text.end_tag
current_color = self.text_colors['default']
scale_override = 1
appear_sequence = None # gets created when calling appear()
text
color                 # sets the default color.
width                 # gets the width of the widest line.
height                # gets the height of the text
lines
wordwrap              # set this to make the text wrap after a certain
number of characters.
background

create_text_section(text, tag='', x=0, y=0)
```

```
align()
create_background(padding=size*2, radius=size,
color=ursina.color.black66)
appear(speed=.025, delay=0)
get_width(string, font=None)
```

```
from ursina import *
  app = Ursina()
  descr = dedent('''
      <scale:1.5><orange>Rainstorm<default><scale:1>
      Summon a <azure>rain storm <default>to deal 5 <azure>water

      damage <default>to <hsb(0,1,.7)>everyone, <default>
<image:file_icon> <red><image:file_icon> test <default>including
<orange>yourself. <default>
      Lasts for 4 rounds.''').strip()

  Text.default_resolution = 1080 * Text.size
  test = Text(text=descr, origin=(0,0), background=True)




  def input(key):
      if key == 'a':
          print('a')
          test.text = '<default><image:file_icon> <red>
<image:file_icon> test '
          print('by', test.text)

  window.fps_counter.enabled = False
  print('....', Text.get_width('yolo'))
  app.run()
```

Button(Entity)

ursina.prefabs.button

Button(text='', radius=.1, **kwargs)

```
parent = camera.ui
collider = 'box'
disabled = False
color = Button.color
text_entity = None
highlight_color = self.color.tint(.2)
pressed_color = self.color.tint(-.2)
highlight_scale = 1    # multiplier
pressed_scale = 1     # multiplier
original_scale = self.scale
icon = None
text
text_origin
text_color

input(key)
on_mouse_enter()
on_mouse_exit()
on_click()
fit_to_text(radius=.1)
```

```
b = Button(text='hello world!', color=color.azure, icon='sword',
scale=.25, text_origin=(-.5,0))
b.on_click = application.quit # assign a function to the button.
b.tooltip = Tooltip('exit')
```

mouse

ursina.mouse

```
enabled = False
visible = True
locked = False
position = Vec3(0,0,0)
delta = Vec3(0,0,0)
```

```
prev_x = 0
prev_y = 0
start_x = 0
start_y = 0
velocity = Vec3(0,0,0)
prev_click_time = time.time()
prev_click_pos = None
double_click_distance = .5
double_click_movement_limit = .01
hovered_entity = None # returns the closest hovered entity with a
collider.
left = False
right = False
middle = False
delta_drag = Vec3(0,0,0)
update_step = 1
traverse_target = scene  # set this to None to disable collision with
scene, which might be a good idea if you have lots of colliders.
raycast = True
collision = None
collisions = list()
enabled = True
x
y
normal             # returns the normal of the polygon, in local
space.
world_normal       # returns the normal of the polygon, in world
space.
point              # returns the point hit, in local space
world_point        # returns the point hit, in world space

input(key)
update()
find_collision()
unhover_everything_not_hit()
```

```
Button(parent=scene, text='a')

def update():
    print(mouse.position, mouse.point)

Cursor()
mouse.visible = False
```

raycaster

ursina.raycaster

```
distance(a, b)
raycast(origin, direction=(0,0,1), distance=inf,
traverse_target=scene, ignore=list(), debug=False)
boxcast(origin, direction=(0,0,1), distance=9999, thickness=(1,1),
traverse_target=scene, ignore=list(), debug=False)   # similar to
raycast, but with width and height
```

```
'''
Casts a ray from *origin*, in *direction*, with length *distance* and
returns
a HitInfo containing information about what it hit. This ray will
only hit entities with a collider.

Use optional *traverse_target* to only be able to hit a specific
entity and its children/descendants.
Use optional *ignore* list to ignore certain entities.
Setting debug to True will draw the line on screen.

Example where we only move if a wall is not hit:
'''


class Player(Entity):

    def update(self):
        self.direction = Vec3(
            self.forward * (held_keys['w'] - held_keys['s'])
            + self.right * (held_keys['d'] - held_keys['a'])
            ).normalized()  # get the direction we're trying to walk
```

```
in.

        origin = self.world_position + (self.up*.5) # the ray should
start slightly up from the ground so we can walk up slopes or walk
over small objects.
        hit_info = raycast(origin , self.direction, ignore=(self,),
distance=.5, debug=False)
        if not hit_info.hit:
            self.position += self.direction * 5 * time.dt

Player(model='cube', origin_y=-.5, color=color.orange)
wall_left = Entity(model='cube', collider='box', scale_y=3,
origin_y=-.5, color=color.azure, x=-4)
wall_right = duplicate(wall_left, x=4)
camera.y = 2
```

application

ursina.application

```
paused = False
time_scale = 1
sequences = list()
trace_entity_definition = False # enable to set entity.line_definition
package_folder = Path(__file__).parent
blender_paths = dict()
development_mode = True
scenes_folder = asset_folder / 'scenes/'
scripts_folder = asset_folder / 'scripts/'
fonts_folder = asset_folder / 'fonts/'
compressed_textures_folder = asset_folder / 'textures_compressed/'
compressed_models_folder = asset_folder / 'models_compressed/'
base = None             # this will be set once the Ursina() is
created
hot_reloader = None     # will be set my main if development_mode

pause()
resume()
quit()
load_settings(path=asset_folder / 'settings.py')
```

build

ursina.__init__

python -m ursina.build

```
open cmd at your project folder and run 'python -m ursina.build' to
package your app for windows.
```

camera

ursina.camera

```
parent = scene
name = 'camera'
eternal = True
ui_size = 40
ui = None
fov = 40
orthographic = False
clip_plane_near
clip_plane_far
aspect_ratio
shader

set_up()
set_shader_input(name, value)
```

```
window.borderless = False

camera.orthographic = True

e = Entity()
e.model = 'quad'
e.color = color.random_color()
e.position = (-2, 0, 10)

e = Entity()
e.model = 'quad'
e.color = color.random_color()
e.position = (2, 0, 10)

e = Entity()
e.model = 'quad'
e.color = color.random_color()
e.position = (0, 0, 40)

EditorCamera()
from ursina.shaders import camera_grayscale_shader
camera.shader = camera_grayscale_shader
```

color

ursina.color

```
color = hsv
white =         color(0, 0, 1)
smoke =         color(0, 0, 0.96)
light_gray =    color(0, 0, 0.75)
gray =          color(0, 0, 0.5)
dark_gray =     color(0, 0, 0.25)
black =         color(0, 0, 0)
red =           color(0, 1, 1)
yellow =        color(60, 1, 1)
lime =          color(90, 1, 1)
green =         color(120, 1, 1)
turquoise =     color(150, 1, 1)
cyan =          color(180, 1, 1)
azure =         color(210, 1, 1)
blue =          color(240, 1, 1)
violet =        color(270, 1, 1)
magenta =       color(300, 1, 1)
pink =          color(330, 1, 1)
brown =         rgb(165, 42, 42)
olive =         rgb(128, 128, 0)
peach =         rgb(255, 218, 185)
gold =          rgb(255, 215, 0)
salmon =        rgb(250, 128, 114)
clear =         Color(0, 0, 0, 0)
white10 =       Color(1,1,1, 0.10)
white33 =       Color(1,1,1, 0.33)
white50 =       Color(1,1,1, 0.50)
white66 =       Color(1,1,1, 0.66)
black10 =       Color(0,0,0, 0.10)
black33 =       Color(0,0,0, 0.33)
black50 =       Color(0,0,0, 0.50)
black66 =       Color(0,0,0, 0.66)
black90 =       Color(0,0,0, 0.90)
text = smoke
light_text = smoke
dark_text = color(0, 0, .1)
text_color = light_text
color_names = ('white', 'smoke', 'light_gray', 'gray', 'dark_gray',
'black',
colors = dict()

hsv(h, s, v, a=1)
rgba(r, g, b, a=255)
rgb(r, g, b, a=255)
to_hsv(color)
hex(value)
brightness(color)
inverse(color)
random_color()
tint(color, amount=.2)
```

```
print(color.brightness(color.blue))
print(_3)

p = Entity(x=-2)
for key in color.colors:
    print(key)
    b = Button(parent=p, model=Quad(subdivisions=2),
color=color.colors[key], text=key)
    b.text_entity.scale *= .5

grid_layout(p.children, max_x=8)

for name in ('r', 'g', 'b', 'h', 's', 'v', 'brightness'):
    print(name + ':', getattr(color.random_color(), name))

e = Entity(model='cube', color=color.lime)
print(e.color.name)
```

curve

ursina.curve

```
linear(t)
in_sine(t)
out_sine(t)
in_out_sine(t)
in_quad(t)
out_quad(t)
in_out_quad(t)
in_cubic(t)
out_cubic(t)
in_out_cubic(t)
in_quart(t)
out_quart(t)
in_out_quart(t)
in_quint(t)
out_quint(t)
in_out_quint(t)
in_expo(t)
out_expo(t)
in_out_expo(t)
in_circ(t)
out_circ(t)
in_out_circ(t)
in_back(t, magnitude=1.70158)
out_back(t, magnitude=1.70158)
in_out_back(t, magnitude=1.70158)
in_elastic(t, magnitude=.7)
out_elastic(t, magnitude=.7)
in_out_elastic(t, magnitude=0.65)
out_bounce(t)
in_bounce(t)
in_out_bounce(t)
{e}_boomerang(t)
```

```
'''Draws a sheet with every curve and its name'''

camera.orthographic = True
camera.fov = 16
camera.position = (9, 6)
window.color = color.black

i = 0
for e in dir(curve):
    try:
        item = getattr(curve, e)
        print(item.__name__, ':', item(.75))
        curve_renderer = Entity(
            model=Mesh(vertices=[Vec3(i / 31, item(i / 31), 0) for i
in range(32)], mode='line', thickness=2),
            color=color.light_gray)
        row = floor(i / 8)
        curve_renderer.x = (i % 8) * 2.5
        curve_renderer.y = row * 1.75
        label = Text(parent=curve_renderer, text=item.__name__,
scale=8, color=color.gray, y=-.1)
        i += 1
    except:
        pass

c = CubicBezier(0, .5, 1, .5)
print('-----------', c.calculate(.23))

window.exit_button.visible = False
window.fps_counter.enabled = False
'''
These are used by Entity when animating, like this:

e = Entity()
e.animate_y(1, curve=curve.in_expo)

e2 = Entity(x=1.5)
e2.animate_y(1, curve=curve.CubicBezier(0,.7,1,.3))
'''
```

duplicate

ursina.duplicate

duplicate(entity, copy_children=True, **kwargs): # use a for loop instead of duplicate()   # use a for loop instead of duplicate() if you can.

```
e = Button(parent=scene, scale=1, text='yolo')
e2 = duplicate(e, x=1.25)
EditorCamera()
```

input_handler

ursina.input_handler

held_keys = defaultdict(lambda: 0)
rebinds = dict()

bind(original_key, alternative_key)
unbind(key)
rebind(to_key, from_key)
input(key)

```
input_handler.bind('s', 'arrow down')  # 's'-key will now be registered as 'arrow down'-key

def test():
    print('----')
def input(key):
    print(key)
    if key == 'left mouse down':
        print('pressed left mouse button')

    if key == Keys.left_mouse_down:   # same as above, but with Keys enum.
        print('pressed left mouse button')


def update():
    for key, value in held_keys.items():
        if value != 0:
            print(key, value)
```

main

ursina.main

time.dt = 0

```
app = Ursina()
app.run()
```

mesh_importer

ursina.mesh_importer

```
blender_scenes = dict()

load_model(name, path=application.asset_folder, file_types=('.bam',
'.ursinamesh', '.obj', '.glb', '.gltf', '.blend'), use_deepcopy=False)
load_blender_scene(name, path=application.asset_folder, load=True,
reload=False, skip_hidden=True, models_only=False)
get_blender(blend_file)   # try to get a matching blender version in
case we have multiple blender version installed
compress_models(path=None,
outpath=application.compressed_models_folder, name='*')
obj_to_ursinamesh()
compress_models_fast(model_name=None, write_to_disk=False)
ursina_mesh_to_obj(mesh, name='',
out_path=application.compressed_models_folder, max_decimals=3)
compress_internal()
```

```
print('imported_meshes:\n', imported_meshes)

t = time.time()

application.asset_folder = application.asset_folder.parent /
'samples'
Entity(model='race')
Entity(model='ambulance', x=1.5)



EditorCamera()
Sky(texture='sky_sunset')
```

scene

ursina.scene

```
render = None
world = None
camera = None
ui_camera = None
entities = []
hidden = NodePath('hidden')
reflection_map_name = 'reflection_map_3'
fog_color
fog_density

set_up()
clear()
```

```
e = Entity(model='plane', color=color.black, scale=100)
EditorCamera()
s = Sky()

def input(key):
    if key == 'l':
        for e in scene.entities:
            print(e.name)

    if key == 'd':
        scene.clear()

scene.fog_density = .1          # sets exponential density
scene.fog_density = (50, 200)   # sets linear density start and end
```

shader

ursina.shader

```
uniform mat4 p3d_ModelViewProjectionMatrix;
```

```
out vec2 uv;
void main() {
}
uniform sampler2D tex;
out vec4 color;
void main() {
}
```

```
from time import perf_counter
t = perf_counter()
Entity(model='cube', shader=Shader())
EditorCamera()
print('ttttttttttttt', perf_counter() - t)
def input(key):
    if held_keys['control'] and key == 'r':
        reload_shaders()

def reload_shaders():
    for e in scene.entities:
        if hasattr(e, '_shader'):
            print('-------', e.shader)
```

string_utilities

ursina.string_utilities

```
camel_to_snake(value)
snake_to_camel(value)
multireplace(string, replacements, ignore_case=False)
printvar(var)
print_info(str, *args)
print_warning(str, *args)
```

```
print(camel_to_snake('CamelToSnake'))
print(snake_to_camel('snake_to_camel'))
printvar('test')
```

text

ursina.text

```
get_width(string, font=None)
```

```
from ursina import *
  app = Ursina()
  descr = dedent('''
      <scale:1.5><orange>Rainstorm<default><scale:1>
      Summon a <azure>rain storm <default>to deal 5 <azure>water

      damage <default>to <hsb(0,1,.7)>everyone, <default>
<image:file_icon> <red><image:file_icon> test <default>including
<orange>yourself. <default>
      Lasts for 4 rounds.''').strip()

  Text.default_resolution = 1080 * Text.size
  test = Text(text=descr, origin=(0,0), background=True)




  def input(key):
      if key == 'a':
          print('a')
          test.text = '<default><image:file_icon> <red>
<image:file_icon> test '
          print('by', test.text)

  window.fps_counter.enabled = False
  print('....', Text.get_width('yolo'))
  app.run()
```

texture_importer
ursina.texture_importer

```
file_types = ('.tif', '.jpg', '.jpeg', '.png', '.gif')
textureless = False

load_texture(name, path=None)
compress_textures(name='')
```

```
Entity(model='quad', texture='white_cube')
```

ursinamath
ursinamath

```
distance(a, b)
distance_2d(a, b)
distance_xz(a, b)
lerp(a, b, t)
inverselerp(a, b, t)
slerp(q1, q2, t)
clamp(value, floor, ceiling)
round_to_closest(value, step=0)
chunk_list(l, chunk_size)
size_list()
sum(l)
```

```
e1 = Entity(position = (0,0,0))
e2 = Entity(position = (0,1,1))
distance(e1, e2)
distance_xz(e1, e2.position)

between_color = lerp(color.lime, color.magenta, .5)
print(between_color)
print(lerp((0,0), (0,1), .5))
print(lerp(Vec2(0,0), Vec2(0,1), .5))
print(lerp([0,0], [0,1], .5))

print(round(Vec3(.38, .1351, 353.26), 2))
```

ursinastuff
ursinastuff

```
invoke(function, *args, **kwargs)
destroy(entity, delay=0)
find_sequence(name, file_types, folders)   # find frame_0, frame_1,
frame_2 and so on
import_all_classes(path=application.asset_folder, debug=False)
print_on_screen(text, position=window.top_left, origin=(-.5,.5),
scale=1, duration=1)
```

```
def test_func(item, x=None, y=None):
    print(item, x, y)

test_func('test')
invoke(test_func, 'test', delay=.1)
invoke(test_func, 'test1', 1, 2, delay=.2)
invoke(test_func, 'test2', x=1, y=2, delay=.3)

def input(key):
    if key == 'space':
        print_on_screen('debug message', position=(0,0), origin=
(0,0), scale=2)
```

window

ursina.window

```
vsync = True   # can't be set during play
show_ursina_splash = False
title = application.asset_folder.name
fullscreen_size = Vec2(*self.screen_resolution)
windowed_size = self.fullscreen_size / 1.25
windowed_position = None   # gets set when entering fullscreen so
position will be correct when going back to windowed mode
forced_aspect_ratio = None # example: window.forced_aspect_ratio
size = self.windowed_size
borderless = True
top = Vec2(0, .5)
bottom = Vec2(0, -.5)
center = Vec2(0, 0)
color = color.dark_gray
render_modes = ('default', 'wireframe', 'colliders', 'normals')
render_mode = 'default'
editor_ui = None
left
right
top_left
top_right
bottom_left
bottom_right
position
icon

late_init()
center_on_screen()
make_editor_gui()   # called by main after setting up camera and
application.development_mode
update_aspect_ratio()
next_render_mode()
```

```
app = Ursina(vsync=True)

window.title = 'ursina'

camera.orthographic = True
camera.fov = 2
```

Audio(Entity)

ursina.audio

Audio(sound_file_name='', autoplay=True, auto_destroy=False, **kwargs)

```
volume = 1
pitch = 1
balance = 0
loop = False
loops = 1
autoplay = autoplay
auto_destroy = auto_destroy
clip
length
status
ready
playing
time

play(start=0)
pause()
resume()
stop(destroy=True)
fade(value, duration=.5, delay=0, curve=curve.in_expo,
resolution=None, interrupt=True)
fade_in(value=1, duration=.5, delay=0, curve=curve.in_expo,
resolution=None, interrupt='finish',)
fade_out(value=0, duration=.5, delay=0, curve=curve.in_expo,
resolution=None, interrupt='finish',)
```

```
from ursina import Ursina, printvar

a = Audio('life_is_currency', pitch=1, loop=True, autoplay=True)
print(a.clip)
a.volume=0
```

```
b = Audio(a.clip)

def input(key):
    if key == 'f':
        a.fade_out(duration=4, curve=curve.linear)

def update():
    print(a.time)
```

Collider()

ursina.collider

Collider()

visible

show()
hide()
remove()

```
e = Entity(model='sphere', x=2)
e.collider = 'box'      # add BoxCollider based on entity's bounds.
e.collider = 'sphere'   # add SphereCollider based on entity's
bounds.
e.collider = 'mesh'     # add MeshCollider based on entity's bounds.

e.collider = BoxCollider(e, center=Vec3(0,0,0), size=Vec3(1,1,1))   #
add BoxCollider at custom positions and size.
e.collider = SphereCollider(e, center=Vec3(0,0,0), radius=.75)      #
add SphereCollider at custom positions and size.
e.collider = MeshCollider(e, mesh=e.model, center=Vec3(0,0,0))      #
add MeshCollider with custom shape and center.

m = Prismatoid(base_shape=Circle(6), thicknesses=(1, .5))
e = Button(parent=scene, model=m, collider='mesh', color=color.red,
highlight_color=color.yellow)

EditorCamera()
```

BoxCollider(Collider)

ursina.collider

BoxCollider(entity, center=(0,0,0), size=(1,1,1))

shape = CollisionBox(Vec3(center[0], center[1], center[2]), size[0],
size[1], size[2])
collision_node = CollisionNode('CollisionNode')
node_path = entity.attachNewNode(self.collision_node)
visible = False

```
e = Entity(model='sphere', x=2)
e.collider = 'box'      # add BoxCollider based on entity's bounds.
e.collider = 'sphere'   # add SphereCollider based on entity's
bounds.
e.collider = 'mesh'     # add MeshCollider based on entity's bounds.

e.collider = BoxCollider(e, center=Vec3(0,0,0), size=Vec3(1,1,1))   #
add BoxCollider at custom positions and size.
e.collider = SphereCollider(e, center=Vec3(0,0,0), radius=.75)      #
add SphereCollider at custom positions and size.
e.collider = MeshCollider(e, mesh=e.model, center=Vec3(0,0,0))      #
add MeshCollider with custom shape and center.

m = Prismatoid(base_shape=Circle(6), thicknesses=(1, .5))
e = Button(parent=scene, model=m, collider='mesh', color=color.red,
highlight_color=color.yellow)

EditorCamera()
```

SphereCollider(Collider)

ursina.collider

SphereCollider(entity, center=(0,0,0), radius=.5)

```
shape = CollisionSphere(center[0], center[1], center[2], radius)
node_path = entity.attachNewNode(CollisionNode('CollisionNode'))
visible = False
```

```
e = Entity(model='sphere', x=2)
e.collider = 'box'      # add BoxCollider based on entity's bounds.
e.collider = 'sphere'   # add SphereCollider based on entity's
bounds.
e.collider = 'mesh'     # add MeshCollider based on entity's bounds.

e.collider = BoxCollider(e, center=Vec3(0,0,0), size=Vec3(1,1,1))   #
add BoxCollider at custom positions and size.
e.collider = SphereCollider(e, center=Vec3(0,0,0), radius=.75)      #
add SphereCollider at custom positions and size.
e.collider = MeshCollider(e, mesh=e.model, center=Vec3(0,0,0))      #
add MeshCollider with custom shape and center.

m = Prismatoid(base_shape=Circle(6), thicknesses=(1, .5))
e = Button(parent=scene, model=m, collider='mesh', color=color.red,
highlight_color=color.yellow)

EditorCamera()
```

MeshCollider(Collider)

ursina.collider

MeshCollider(entity, mesh=None, center=(0,0,0))

```
node_path = entity.attachNewNode(CollisionNode('CollisionNode'))
node = self.node_path.node()
collision_polygons = list()
visible = False
```

```
e = Entity(model='sphere', x=2)
e.collider = 'box'      # add BoxCollider based on entity's bounds.
e.collider = 'sphere'   # add SphereCollider based on entity's
bounds.
e.collider = 'mesh'     # add MeshCollider based on entity's bounds.

e.collider = BoxCollider(e, center=Vec3(0,0,0), size=Vec3(1,1,1))   #
add BoxCollider at custom positions and size.
e.collider = SphereCollider(e, center=Vec3(0,0,0), radius=.75)      #
add SphereCollider at custom positions and size.
e.collider = MeshCollider(e, mesh=e.model, center=Vec3(0,0,0))      #
add MeshCollider with custom shape and center.

m = Prismatoid(base_shape=Circle(6), thicknesses=(1, .5))
e = Button(parent=scene, model=m, collider='mesh', color=color.red,
highlight_color=color.yellow)

EditorCamera()
```

CollisionZone(Entity)

ursina.collision_zone

CollisionZone(radius=2, target_entities=None, **kwargs)

```
radius = radius
entities_with_mesh_colliders = target_entities # defaults to all
entities with a mesh collider

update()
update_colliders()
```

```
'''
This will only enable mesh colliders' collision polygons within a
certain range,
in order to improve performance.
'''

from ursina.shaders import basic_lighting_shader
window.vsync = False

application.asset_folder = application.asset_folder.parent
terrain = Entity(model='heightmap_test', scale=32, texture='grass',
collider='mesh', shader=basic_lighting_shader)
from ursina.prefabs.first_person_controller import
FirstPersonController
player = FirstPersonController(speed=10)
collision_zone = CollisionZone(parent=player, radius=32)


def input(key):
    if key == 'c':
        terrain.collision = not terrain.collision



Sky()
base.set_frame_rate_meter(True)
```

Color(Vec4)

ursina.color

Color(self,*p)

name
r
g
b
a
hsv
h
s
v
brightness

invert()
tint(amount)
hsv(h, s, v, a=1)
rgba(r, g, b, a=255)
rgb(r, g, b, a=255)
to_hsv(color)
hex(value)
brightness(color)
inverse(color)
random_color()
tint(color, amount=.2)

```
print(color.brightness(color.blue))
print(_3)

p = Entity(x=-2)
for key in color.colors:
    print(key)
    b = Button(parent=p, model=Quad(subdivisions=2),
color=color.colors[key], text=key)
    b.text_entity.scale *= .5

grid_layout(p.children, max_x=8)

for name in ('r', 'g', 'b', 'h', 's', 'v', 'brightness'):
    print(name + ':', getattr(color.random_color(), name))

e = Entity(model='cube', color=color.lime)
print(e.color.name)
```

CubicBezier

ursina.curve

CubicBezier(a, b, c, d)

```
a = a
b = b
c = c
d = d
cx = 3.0 * a
bx = 3.0 * (c - a) - self.cx
ax = 1.0 - self.cx - self.bx
cy = 3.0 * b
by = 3.0 * (d - b) - self.cy
ay = 1.0 - self.cy - self.by

sample_curve_x(t)
sample_curve_y(t)
sample_curve_derivative_x(t)
calculate(x, epsilon=.0001)
solve_curve_x(t, epsilon=.0001)
```

```
'''Draws a sheet with every curve and its name'''

camera.orthographic = True
camera.fov = 16
camera.position = (9, 6)
window.color = color.black

i = 0
for e in dir(curve):
    try:
        item = getattr(curve, e)
        print(item.__name__, ':', item(.75))
        curve_renderer = Entity(
            model=Mesh(vertices=[Vec3(i / 31, item(i / 31), 0) for i
in range(32)], mode='line', thickness=2),
            color=color.light_gray)
        row = floor(i / 8)
        curve_renderer.x = (i % 8) * 2.5
        curve_renderer.y = row * 1.75
        label = Text(parent=curve_renderer, text=item.__name__,
scale=8, color=color.gray, y=-.1)
        i += 1
    except:
        pass

c = CubicBezier(0, .5, 1, .5)
print('-----------', c.calculate(.23))

window.exit_button.visible = False
window.fps_counter.enabled = False
'''
These are used by Entity when animating, like this:

e = Entity()
e.animate_y(1, curve=curve.in_expo)

e2 = Entity(x=1.5)
e2.animate_y(1, curve=curve.CubicBezier(0,.7,1,.3))
'''
```

HitInfo

ursina.hit_info

HitInfo(kwargs)**

```
hit = None
entity = None
point = None
world_point = None
distance = math.inf
normal = None
world_normal = None
hits = []
entities = []
```

Keys(Enum)

ursina.input_handler

```
left_mouse_down = 'left mouse down'
left_mouse_up = 'left mouse up'
middle_mouse_down = 'middle mouse down'
middle_mouse_up = 'middle mouse up'
right_mouse_down = 'right mouse down'
right_mouse_up = 'right mouse up'
scroll_up = 'scroll up'
scroll_down = 'scroll down'
left_arrow = 'left arrow'
left_arrow_up = 'left arrow up'
up_arrow = 'up arrow'
up_arrow_up = 'up arrow up'
down_arrow = 'down arrow'
down_arrow_up = 'down arrow up'
right_arrow = 'right arrow'
right_arrow_up = 'right arrow up'
left_control = 'left control'
right_control = 'right control'
left_shift = 'left shift'
right_shift = 'right shift'
left_alt = 'left alt'
right_alt = 'right alt'
left_control_up = 'left control up'
right_control_up = 'right control up'
left_shift_up = 'left shift up'
right_shift_up = 'right shift up'
left_alt_up = 'left alt up'
right_alt_up = 'right alt up'
page_down = 'page down'
page_down_up = 'page down up'
page_up = 'page up'
page_up_up = 'page up up'
enter = 'enter'
backspace = 'backspace'
escape = 'escape'
tab = 'tab'
gamepad_a = 'gamepad a'
gamepad_a_up = 'gamepad a up'
gamepad_b = 'gamepad b'
gamepad_b_up = 'gamepad b up'
gamepad_x = 'gamepad x'
gamepad_x_up = 'gamepad x up'
gamepad_y = 'gamepad y'
gamepad_y_up = 'gamepad y up'
gamepad_left_stick = 'gamepad left stick'
gamepad_left_stick_up = 'gamepad left stick up'
gamepad_right_stick = 'gamepad right stick'
gamepad_right_stick_up = 'gamepad right stick up'
gamepad_back = 'gamepad back'
gamepad_back_up = 'gamepad back up'
gamepad_start = 'gamepad start'
gamepad_dpad_down = 'gamepad dpad down'
gamepad_dpad_down_up = 'gamepad dpad down up'
gamepad_dpad_up = 'gamepad dpad up'
gamepad_dpad_up_up = 'gamepad dpad up up'
gamepad_dpad_left = 'gamepad dpad left'
gamepad_dpad_left_up = 'gamepad dpad left up'
gamepad_dpad_right = 'gamepad dpad right'
gamepad_dpad_right_up = 'gamepad dpad right up'
gamepad_left_shoulder = 'gamepad left shoulder'
gamepad_left_shoulder_up = 'gamepad left shoulder up'
gamepad_right_shoulder = 'gamepad right shoulder'
gamepad_right_shoulder_up = 'gamepad right shoulder up'
```

Light(Entity)

ursina.lights

Light(kwargs)**

color

```
from ursina.shaders import lit_with_shadows_shader # you have to apply this shader to enties for them to receive shadows.
EditorCamera()
Entity(model='plane', scale=10, color=color.gray, shader=lit_with_shadows_shader)
Entity(model='cube', y=1, shader=lit_with_shadows_shader)
pivot = Entity()
DirectionalLight(parent=pivot, y=2, z=3, shadows=True)
```

DirectionalLight(Light)

ursina.lights

DirectionalLight(shadows=True, **kwargs)

shadow_map_resolution = Vec2(1024, 1024)
shadows

```
from ursina.shaders import lit_with_shadows_shader # you have to apply this shader to enties for them to receive shadows.
EditorCamera()
Entity(model='plane', scale=10, color=color.gray, shader=lit_with_shadows_shader)
Entity(model='cube', y=1, shader=lit_with_shadows_shader)
pivot = Entity()
DirectionalLight(parent=pivot, y=2, z=3, shadows=True)
```

PointLight(Light)

ursina.lights

PointLight(kwargs)**

```
from ursina.shaders import lit_with_shadows_shader # you have to apply this shader to enties for them to receive shadows.
EditorCamera()
Entity(model='plane', scale=10, color=color.gray, shader=lit_with_shadows_shader)
Entity(model='cube', y=1, shader=lit_with_shadows_shader)
pivot = Entity()
DirectionalLight(parent=pivot, y=2, z=3, shadows=True)
```

AmbientLight(Light)

ursina.lights

AmbientLight(kwargs)**

```
from ursina.shaders import lit_with_shadows_shader # you have to apply this shader to enties for them to receive shadows.
EditorCamera()
Entity(model='plane', scale=10, color=color.gray, shader=lit_with_shadows_shader)
Entity(model='cube', y=1, shader=lit_with_shadows_shader)
pivot = Entity()
DirectionalLight(parent=pivot, y=2, z=3, shadows=True)
```

SpotLight(Light)

ursina.lights

SpotLight(kwargs)**

```
from ursina.shaders import lit_with_shadows_shader # you have to apply this shader to enties for them to receive shadows.
EditorCamera()
Entity(model='plane', scale=10, color=color.gray, shader=lit_with_shadows_shader)
Entity(model='cube', y=1, shader=lit_with_shadows_shader)
pivot = Entity()
DirectionalLight(parent=pivot, y=2, z=3, shadows=True)
```

Ursina()

ursina.main

Ursina(kwargs): # optional arguments: title, fullscreen, size, forced_aspect_ratio, position, vsync, borderless, show_ursina_splash, render_mode, development_mode, editor_ui_enabled**

```
mouse = mouse

input_up(key)
input_hold(key)
input(key)
run()
```

```
app = Ursina()
app.run()
```

MeshModes(Enum)

ursina.mesh

```
triangle = 'triangle'
ngon = 'ngon'
quad = 'quad'
line = 'line'
point = 'point'
tristrip = 'tristrip'
```

Mesh()

ursina.mesh

Mesh(vertices=None, triangles=None, colors=None, uvs=None, normals=None, static=True, mode='triangle', thickness=1, render_points_in_3d=True)

```
name = 'mesh'
vertices = vertices
triangles = triangles
colors = colors
uvs = uvs
normals = normals
static = static
mode = mode
thickness = thickness
render_points_in_3d = render_points_in_3d
recipe

generate()   # call this after setting some of the variables to update it
generate_normals(smooth=True)
colorize(left=color.white, right=color.blue, down=color.red, up=color.green, back=color.white, forward=color.white, smooth=True, world_space=True, strength=1)
project_uvs(aspect_ratio=1, direction='forward')
```

```
clear(regenerate=True)
save(name='', path=application.compressed_models_folder)
```

```
verts = ((0,0,0), (1,0,0), (.5, 1, 0), (-.5,1,0))
tris = (1, 2, 0, 2, 3, 0)
uvs = ((1.0, 0.0), (0.0, 1.0), (0.0, 0.0), (1.0, 1.0))
norms = ((0,0,-1),) * len(verts)
colors = (color.red, color.blue, color.lime, color.black)


e = Entity(model=Mesh(vertices=verts, triangles=tris, uvs=uvs,
normals=norms, colors=colors), scale=2)
verts = (Vec3(0,0,0), Vec3(0,1,0), Vec3(1,1,0), Vec3(2,2,0),
Vec3(0,3,0), Vec3(-2,3,0))
tris = ((0,1), (3,4,5))

lines = Entity(model=Mesh(vertices=verts, triangles=tris,
mode='line', thickness=4), color=color.cyan, z=-1)
points = Entity(model=Mesh(vertices=verts, mode='point',
thickness=.05), color=color.red, z=-1.01)
```

Wait()

ursina.sequence

Wait(duration)

```
duration = duration
```

```
e = Entity(model='quad')
s = Sequence(
    1,
    Func(print, 'one'),
    Func(e.fade_out, duration=1),
    1,
    Func(print, 'two'),
    Func(e.fade_in, duration=1),
    loop=True
    )

s.append(
    Func(print, 'appended to sequence')
    )

def input(key):
    actions = {'s' : s.start, 'f' : s.finish, 'p' : s.pause, 'r' :
s.resume}
    if key in actions:
        actions[key]()
```

Func()

ursina.sequence

Func(func, *args, **kwargs)

```
func = func
args = args
kwargs = kwargs
delay = 0
finished = False
```

```
e = Entity(model='quad')
s = Sequence(
    1,
    Func(print, 'one'),
    Func(e.fade_out, duration=1),
    1,
    Func(print, 'two'),
    Func(e.fade_in, duration=1),
    loop=True
    )
```

```
s.append(
    Func(print, 'appended to sequence')
    )

def input(key):
    actions = {'s' : s.start, 'f' : s.finish, 'p' : s.pause, 'r' :
s.resume}
    if key in actions:
        actions[key]()
```

Sequence()

ursina.sequence

Sequence(*args, **kwargs)

```
args = list(args)
t = 0
time_step = Sequence.default_time_step
duration = 0
funcs = list()
paused = True
loop = False
auto_destroy = True
finished

generate()
append(arg)
extend(list)
start()
pause()
resume()
finish()
kill()
update()
```

```
e = Entity(model='quad')
s = Sequence(
    1,
    Func(print, 'one'),
    Func(e.fade_out, duration=1),
    1,
    Func(print, 'two'),
    Func(e.fade_in, duration=1),
    loop=True
    )

s.append(
    Func(print, 'appended to sequence')
    )

def input(key):
    actions = {'s' : s.start, 'f' : s.finish, 'p' : s.pause, 'r' :
s.resume}
    if key in actions:
        actions[key]()
```

Shader

ursina.shader

Shader(language=Panda3dShader.SL_GLSL, vertex=default_vertex_shader, fragment=default_fragment_shader, geometry='', **kwargs)

```
path = Path(_caller.filename)
language = language
vertex = vertex
fragment = fragment
geometry = geometry
entity = None
default_input = dict()
compiled = False

compile()
```

```
from time import perf_counter
t = perf_counter()
Entity(model='cube', shader=Shader())
EditorCamera()
print('ttttttttttttt', perf_counter() - t)
def input(key):
    if held_keys['control'] and key == 'r':
        reload_shaders()

def reload_shaders():
    for e in scene.entities:
        if hasattr(e, '_shader'):
            print('-------', e.shader)
```

Texture()

ursina.texture

Texture(value)

```
filtering = Texture.default_filtering      # None/'bilinear'/'mipmap'
default: 'bilinear'
name
size
width
height
pixels
repeat

get_pixel(x, y)
get_pixels(start, end)
set_pixel(x, y, color)
apply()
save(path)
```

```
'''
    The Texture class rarely used manually but usually instantiated
    when assigning a texture to an Entity
    texture = Texture(path / PIL.Image / panda3d.core.Texture)

    A texture file can be a .png, .jpg or .psd.
    If it's a .psd it and no compressed version exists, it will
compress it automatically.
'''
e = Entity(model='quad', texture='brick')
e.texture.set_pixel(0, 2, color.blue)
e.texture.apply()
```

Trigger(Entity)

ursina.trigger

Trigger(kwargs)**

```
trigger_targets = None
radius = .5
triggerers = list()
update_rate = 4

update()
```

```
player = Entity(model='cube', color=color.azure, scale=.05)
def update():
    player.x += held_keys['d'] * time.dt * 2
    player.x -= held_keys['a'] * time.dt * 2

t = Trigger(trigger_targets=(player,), x=1, model='sphere',
color=color.color(0,1,1,.5))
t.on_trigger_enter = Func(print, 'enter')
t.on_trigger_exit =  Func(print, 'exit')
t.on_trigger_stay =  Func(print, 'stay')
```

Empty()

ursinastuff

Empty(kwargs)**

```
invoke(function, *args, **kwargs)
destroy(entity, delay=0)
find_sequence(name, file_types, folders)   # find frame_0, frame_1,
frame_2 and so on
import_all_classes(path=application.asset_folder, debug=False)
print_on_screen(text, position=window.top_left, origin=(-.5,.5),
scale=1, duration=1)
```

```
def test_func(item, x=None, y=None):
    print(item, x, y)

test_func('test')
invoke(test_func, 'test', delay=.1)
invoke(test_func, 'test1', 1, 2, delay=.2)
invoke(test_func, 'test2', x=1, y=2, delay=.3)

def input(key):
    if key == 'space':
        print_on_screen('debug message', position=(0,0), origin=
(0,0), scale=2)
```

LoopingList(list)

ursinastuff

LoopingList(kwargs)**

```
def test_func(item, x=None, y=None):
    print(item, x, y)

test_func('test')
invoke(test_func, 'test', delay=.1)
invoke(test_func, 'test1', 1, 2, delay=.2)
invoke(test_func, 'test2', x=1, y=2, delay=.3)

def input(key):
    if key == 'space':
        print_on_screen('debug message', position=(0,0), origin=
(0,0), scale=2)
```

Vec2(PandaVec2)

ursina.vec2

Vec2(kwargs)**

```
x
y
```

```
a = Vec2(1,1)
print(a)
print(round(a))
```

Vec3(PandaVec3)

ursina.vec3

Vec3(kwargs)**

```
x
y
z
xy
```

```
xz
yz
```

```
a = Vec3(1,0,0) * 2
a = Vec3(1,0,1) * Vec3(2,1,2)
b = Vec3(1.252352324,0,1)
b += Vec3(0,1)
```

Animation(Sprite)

ursina.prefabs.animation

Animation(name, fps=12, loop=True, autoplay=True, frame_times=None, **kwargs)

```
sequence = Sequence(loop=loop, auto_destroy=False)
frame_times = frame_times
is_playing = False
autoplay = autoplay
duration            # get the duration of the animation. you can't set it. to do so, change the fps instead.

start()
pause()
resume()
finish()
```

```
'''
Loads an image sequence as a frame animation.
So if you have some frames named image_000.png, image_001.png, image_002.png and so on,
you can load it like this: Animation('image')

You can also load a .gif by including the file type: Animation('image.gif')
'''

Animation('ursina_wink')
```

Animator()

ursina.prefabs.animator

Animator(animations=None, start_state='')

```
animations = animations     # dict
start_state = start_state
state = start_state
```

```
anim = Animation('ursina_wink', loop=True, autoplay=False)
a = Animator(
    animations = {
        'lol' : Entity(model='cube', color=color.red),
        'yo' : Entity(model='cube', color=color.green, x=1),
        'help' : anim,
    }
)
a.state = 'yo'

Text('press <red>1<default>, <green>2<default> or <violet>3<default> to toggle different animator states', origin=(0,-.5), y=-.4)

def input(key):
    if key == '1':
        a.state = 'lol'
    if key == '2':
        a.state = 'yo'
    if key == '3':
        a.state = 'help'
        print(anim.enabled)
```

ButtonGroup(Entity)

ursina.prefabs.button_group

ButtonGroup(options=None, default='', min_selection=1, max_selection=1, **kwargs)

```
deselected_color = Button.color
selected_color = color.azure
min_selection = min_selection
max_selection = max(min_selection, max_selection)
buttons = list()
selected = list()
options = options
parent = camera.ui
scale = Text.size * 2
value

layout()
input(key)
select(b)
on_value_changed()
```

```
gender_selection = ButtonGroup(('man', 'woman', 'other'))
on_off_switch = ButtonGroup(('off', 'on'), min_selection=1, y=-.1,
default='on', selected_color=color.red)

def on_value_changed():
    print('set gender:', gender_selection.value)
gender_selection.on_value_changed = on_value_changed

def on_value_changed():
    print('turn:', on_off_switch.value)
on_off_switch.on_value_changed = on_value_changed

window.color = color._32
```

ButtonList(Entity)

ursina.prefabs.button_list

ButtonList(button_dict, button_height=1.1, fit_height=True, width=.5, **kwargs)

```
fit_height = fit_height
button_height = button_height
text_entity = Text(parent=self, origin=(-.5,.5), text='empty',
world_scale=20, z=-.1, x=.01, line_height=button_height)
button_height = self.text_entity.height
button_dict = button_dict
highlight = Entity(parent=self, model='quad', color=color.white33,
scale=(1,self.button_height), origin=(-.5,.5), z=-.01,
add_to_scene_entities=False)
selection_marker = Entity(parent=self, model='quad',
color=color.azure, scale=(1,self.button_height), origin=(-.5,.5),
z=-.02, enabled=False, add_to_scene_entities=False)

input(key)
update()
on_disable()
```

```
default = Func(print, 'not yet implemented')

def test(a=1, b=2):
    print('------:', a, b)

button_dict = {
    'one' :     None,
    'two' :     default,
    'tree' :    Func(test, 3, 4),
    'four' :    Func(test, b=3, a=4),
}
for i in range(6, 20):
    button_dict[f'button {i}'] = Func(print, i)

sound_effects = {}
current_sound = None
```

```
for name in ('blip', 'boom', 'coin', 'hurt', 'jump', 'lose',
'powerup', 'teleport'):
    for i in range(1,4):
        a = Audio(f'{name}_{i}', autoplay=False)
        if a.clip:
            sound_effects[f'{name}_{i}'] = Func(print, 'a')

bl = ButtonList(sound_effects)
```

Node

ursina.prefabs.conversation

Node(button_dict, button_height=1.1, fit_height=True, width=.5, **kwargs)

```
app = Ursina()
    conversation = Conversation()
    bar_mission_solved = False
    convo = '''
I'm looking for my sister. Can you help me find her, please? I
haven't seen her in days! Who know what could've happened!?
I'm worried. Will you help me?
    * Yes, of course. This can be a dangerous city.
        Oh no! Do you think something happened to her?
        What should I do?!
            * She's probably fine. She can handle herself.
                You're right. I'm still worried though.
                    * Don't worry, I'll look for her.
            * Maybe. (chaos += 1)
                Help me look for her, please! *runs off*
    * I'm sorry, but I don't have time right now.
        A true friend wouldn't say that. (evil += 1)
    * I know where she is! (if bar_mission_solved)
        Really? Where?
            * I saw her on a ship by the docks, it looked like they
were ready to set off.
                Thank you! *runs off*
'''
    conversation.start_conversation(convo)

    window.size = window.fullscreen_size * .5
    Sprite('shore', z=1)
    app.run()
```

Conversation(Entity)

ursina.prefabs.conversation

Conversation(kwargs)**

```
question = Button(
        parent=self,
        text_origin=(-.5,0),
        scale_x=1,
        scale_y=.1,
        model=Quad(radius=.5, aspect=1/.1),
        text='What do you want\nWhat do you want?'
        )
more_indicator = Entity(parent=self.question, model=Circle(3),
position=(.45,-.4,-.1), rotation_z=180, color=color.azure,
world_scale=.5, z=-1, enabled=False)
spacing = 4 * .02
wordwrap = 65
button_model = Quad(radius=.5, aspect=1/.075)
answer_0 = Button(parent=self, text='answer_0', y=self.question.y-
self.spacing-.025, scale=(1,.075), text_origin=(-.5,0),
model=copy(self.button_model))
answer_1 = Button(parent=self, text='answer_1', y=self.answer_0.y-
self.spacing, scale=(1,.075), text_origin=(-.5,0),
model=copy(self.button_model))
answer_2 = Button(parent=self, text='answer_2', y=self.answer_1.y-
self.spacing, scale=(1,.075), text_origin=(-.5,0),
model=copy(self.button_model))
buttons = (self.answer_0, self.answer_1, self.answer_2)
question_appear_sequence = None
button_appear_sequence = None
started = False

toggle()
ask(node, question_part=0)
on_click(node=child)
input(key)
next()
start_conversation(conversation)
parse_conversation(convo)
```

```
app = Ursina()
    conversation = Conversation()
    bar_mission_solved = False
    convo = '''
I'm looking for my sister. Can you help me find her, please? I
haven't seen her in days! Who know what could've happened!?
I'm worried. Will you help me?
    * Yes, of course. This can be a dangerous city.
        Oh no! Do you think something happened to her?
        What should I do?!
            * She's probably fine. She can handle herself.
                You're right. I'm still worried though.
                    * Don't worry, I'll look for her.
            * Maybe. (chaos += 1)
                Help me look for her, please! *runs off*
    * I'm sorry, but I don't have time right now.
        A true friend wouldn't say that. (evil += 1)
    * I know where she is! (if bar_mission_solved)
        Really? Where?
            * I saw her on a ship by the docks, it looked like they
were ready to set off.
                Thank you! *runs off*
'''
    conversation.start_conversation(convo)

    window.size = window.fullscreen_size * .5
    Sprite('shore', z=1)
    app.run()
```

Cursor(Entity)

ursina.prefabs.cursor

Cursor(kwargs)**

```
parent = camera.ui
texture = 'cursor'
model = 'quad'
color = color.light_gray
render_queue = 1

update()
```

```
Button('button').fit_to_text()
Panel()
camera.orthographic = True
camera.fov = 100
e = Entity(model='cube')
mouse._mouse_watcher.setGeometry(e.model.node())
mouse.visible = False
```

DebugMenu(Draggable)

ursina.prefabs.debug_menu

DebugMenu(target, **kwargs)

```
target = target
scale = (.2, .025)
text = '<orange>' + target.__class__.__name__
```

```
draw_functions()
```

```
DebugMenu(Audio('night_sky'))
```

Draggable(Button)

ursina.prefabs.draggable

Draggable(kwargs)**

```
require_key = None
dragging = False
delta_drag = 0
start_pos = self.world_position
start_offset = (0,0,0)
step = (0,0,0)
plane_direction = (0,0,1)
lock = Vec3(0,0,0)     # set to 1 to lock movement on any of x, y and
z axes
min_x, self.min_y, self.min_z = -inf, -inf, -inf
max_x, self.max_y, self.max_z = inf, inf, inf

input(key)
start_dragging()
stop_dragging()
update()
```

```
Entity(model='plane', scale=8, texture='white_cube', texture_scale=
(8,8))
draggable_button = Draggable(scale=.1, text='drag me', position=(-.5,
0))
world_space_draggable = Draggable(parent=scene, model='cube',
color=color.azure, plane_direction=(0,1,0), lock=(1,0,0))

EditorCamera(rotation=(30,10,0))
world_space_draggable.drop = Func(print, 'dropped cube')
```

DropdownMenuButton(Button)

ursina.prefabs.dropdown_menu

DropdownMenuButton(text='', **kwargs)

```
from ursina.prefabs.dropdown_menu import DropdownMenu,
DropdownMenuButton

DropdownMenu('File', buttons=(
    DropdownMenuButton('New'),
    DropdownMenuButton('Open'),
    DropdownMenu('Reopen Project', buttons=(
        DropdownMenuButton('Project 1'),
        DropdownMenuButton('Project 2'),
        )),
    DropdownMenuButton('Save'),
    DropdownMenu('Options', buttons=(
        DropdownMenuButton('Option a'),
        DropdownMenuButton('Option b'),
        )),
    DropdownMenuButton('Exit'),
    ))
```

DropdownMenu(DropdownMenuButton)

ursina.prefabs.dropdown_menu

DropdownMenu(text='', buttons=list(), **kwargs)

```
position = window.top_left
buttons = buttons
```

arrow_symbol = Text(world_parent=self, text='>', origin=(.5,.5), position=(.95, 0), color=color.gray)

open()
close()
on_mouse_enter()
input(key)
update()

```
from ursina.prefabs.dropdown_menu import DropdownMenu,
DropdownMenuButton

DropdownMenu('File', buttons=(
    DropdownMenuButton('New'),
    DropdownMenuButton('Open'),
    DropdownMenu('Reopen Project', buttons=(
        DropdownMenuButton('Project 1'),
        DropdownMenuButton('Project 2'),
        )),
    DropdownMenuButton('Save'),
    DropdownMenu('Options', buttons=(
        DropdownMenuButton('Option a'),
        DropdownMenuButton('Option b'),
        )),
    DropdownMenuButton('Exit'),
    ))
```

EditorCamera(Entity)

ursina.prefabs.editor_camera

EditorCamera(kwargs)**

gizmo = Entity(parent=self, model='sphere', color=color.orange, scale=.025, add_to_scene_entities=False, enabled=False)
rotation_speed = 200
pan_speed = Vec2(5, 5)
move_speed = 10
zoom_speed = 1.25
zoom_smoothing = 8
rotate_around_mouse_hit = False
start_position = self.position
perspective_fov = camera.fov
orthographic_fov = camera.fov
on_destroy = self.on_disable
hotkeys = {'toggle_orthographic':'shift+p', 'focus':'f', 'reset_center':'shift+f'}

on_enable()
on_disable()
input(key)
update()

```
app = Ursina(vsync=False)
'''
Simple camera for debugging.
Hold right click and move the mouse to rotate around point.
'''

sky = Sky()
e = Entity(model='cube', color=color.white, collider='box')
e.model.colorize()

from ursina.prefabs.first_person_controller import
FirstPersonController

ground = Entity(model='plane', scale=32, texture='white_cube',
texture_scale=(32,32), collider='box')
box = Entity(model='cube', collider='box', texture='white_cube',
scale=(10,2,2), position=(2,1,5), color=color.light_gray)
ec = EditorCamera(rotation_smoothing=2, enabled=1, rotation=
(30,30,0))

rotation_info = Text(position=window.top_left)

def update():
    rotation_info.text = str(int(ec.rotation_y)) + '\n' +
str(int(ec.rotation_x))
```

```
def input(key):
    if key == 'tab':    # press tab to toggle edit/play mode
        ec.enabled = not ec.enabled
        player.enabled = not player.enabled
```

ExitButton(Button)

ursina.prefabs.exit_button

ExitButton(kwargs)**

on_click()
input(key)

```
'''
This is the button in the upper right corner.
You can click on it or press Shift+Q to close the program.
To disable it, set window.exit_button.enabled to False
'''
```

FileButton(Button)

ursina.prefabs.file_browser

FileButton(load_menu, **kwargs)

load_menu = load_menu
selected

on_click()
on_double_click()

```
fb = FileBrowser(file_types=('.*'), enabled=True)

def on_submit(paths):
    print('--------', paths)
    for p in paths:
        print('---', p)

fb.on_submit = on_submit
```

FileBrowser(Entity)

ursina.prefabs.file_browser

FileBrowser(kwargs)**

file_types = ['.*',]
start_path = Path('.').resolve()
return_files = True
return_folders = False
selection_limit = 1
max_buttons = 24
title_bar = Button(parent=self, scale=(.9,.035), text='<gray>Open', color=color.dark_gray, collision=False)
address_bar = Button(parent=self, scale=(.8,.035), text='//', text_origin=(-.5,0), y=-.05, highlight_color=color.black)
folder_up_button = Button(parent=self, scale=(.035,.035), texture='arrow_down', rotation_z=180, position=(-.42,-.05), color=color.white, highlight_color=color.azure, on_click=self.folder_up)
button_parent = Entity(parent=self)
back_panel = Entity(parent=self, model='quad', collider='box', origin_y=.5, scale=(.9,(self.max_buttons*.025)+.19), color=color._32, z=.1)
bg = Button(parent=self, z=1, scale=(999,999), color=color.black66, highlight_color=color.black66, pressed_color=color.black66)
cancel_button = Button(parent=self, scale=(.875*.24, .05), y=(-self.max_buttons*.025)-.15, origin_x=-.5, x=-.875/2, text='Cancel',

```
on_click=self.close)
open_button = Button(parent=self, scale=(.875*.74, .05), y=(-
self.max_buttons*.025)-.15, origin_x=.5, x=.875/2, text='Open',
color=color.dark_gray, on_click=self.open)
cancel_button_2 = Button(parent=self.title_bar, model=Circle(),
world_scale=self.title_bar.world_scale_y*.75, origin_x=.5, x=.495,
z=-.1, text='<gray>x', on_click=self.close)
can_scroll_up_indicator = Entity(parent=self, model='quad',
texture='arrow_down', rotation_z=180, scale=(.05,.05), y=-.0765,
z=-.1, color=color.dark_gray, enabled=False,
add_to_scene_entities=False)
can_scroll_down_indicator = Entity(parent=self, model='quad',
texture='arrow_down', scale=(.05,.05), y=(-
self.max_buttons*.025)-.104, z=-.1, color=color.dark_gray,
enabled=False, add_to_scene_entities=False)
scroll
path
selection

input(key)
on_enable()
close()
folder_up()
open(path=None)
```

```
fb = FileBrowser(file_types=('.*'), enabled=True)

def on_submit(paths):
    print('--------', paths)
    for p in paths:
        print('---', p)

fb.on_submit = on_submit
```

FileBrowserSave(FileBrowser)

ursina.prefabs.file_browser_save

FileBrowserSave(kwargs)**

```
save_button = self.open_button
file_name_field = InputField(
        parent
file_type = '' # to save as
data = ''
last_saved_file = None     # gets set when you save a file
overwrite_prompt = WindowPanel(
        content=(
            Text('Overwrite?'),
            Button('Yes', color=color.azure, on_click=self.save),
            Button('Cancel')
        ),
        z=-1,
        popup=True,
        enabled=False)

save()
```

```
from ursina.prefabs.file_browser_save import FileBrowserSave

wp = FileBrowserSave(file_type = '.oto')


import json
save_data = {'level': 4, 'name':'Link'}
wp.data = json.dumps(save_data)
```

FirstPersonController(Entity)

ursina.prefabs.first_person_controller

FirstPersonController(kwargs)**

```
cursor = Entity(parent=camera.ui, model='quad', color=color.pink,
scale=.008, rotation_z=45)
```

```
speed = 5
height = 2
camera_pivot = Entity(parent=self, y=self.height)
mouse_sensitivity = Vec2(40, 40)
gravity = 1
grounded = False
jump_height = 2
jump_up_duration = .5
fall_after = .35 # will interrupt jump up
jumping = False
air_time = 0

update()
input(key)
jump()
start_fall()
land()
on_enable()
on_disable()
```

```
from ursina.prefabs.first_person_controller import
FirstPersonController
ground = Entity(model='plane', scale=(100,1,100),
color=color.yellow.tint(-.2), texture='white_cube', texture_scale=
(100,100), collider='box')
e = Entity(model='cube', scale=(1,5,10), x=2, y=.01, rotation_y=45,
collider='box', texture='white_cube')
e.texture_scale = (e.scale_z, e.scale_y)
e = Entity(model='cube', scale=(1,5,10), x=-2, y=.01, collider='box',
texture='white_cube')
e.texture_scale = (e.scale_z, e.scale_y)

player = FirstPersonController(y=2, origin_y=-.5)
player.gun = None


gun = Button(parent=scene, model='cube', color=color.blue,
origin_y=-.5, position=(3,0,3), collider='box')
gun.on_click = Sequence(Func(setattr, gun, 'parent', camera),
Func(setattr, player, 'gun', gun))

gun_2 = duplicate(gun, z=7, x=8)
slope = Entity(model='cube', collider='box', position=(0,0,8),
scale=6, rotation=(45,0,0), texture='brick', texture_scale=(8,8))
slope = Entity(model='cube', collider='box', position=(5,0,10),
scale=6, rotation=(80,0,0), texture='brick', texture_scale=(8,8))

hookshot_target = Button(parent=scene, model='cube',
color=color.brown, position=(4,5,5))
hookshot_target.on_click = Func(player.animate_position,
hookshot_target.position, duration=.5, curve=curve.linear)

def input(key):
    if key == 'left mouse down' and player.gun:
        gun.blink(color.orange)
        bullet = Entity(parent=gun, model='cube', scale=.1,
color=color.black)
        bullet.world_parent = scene
        bullet.animate_position(bullet.position+(bullet.forward*50),
curve=curve.linear, duration=1)
        destroy(bullet, delay=1)
```

FrameAnimation3d(Entity)

ursina.prefabs.frame_animation_3d

FrameAnimation3d(name, fps=12, loop=True, autoplay=True, frame_times=None, **kwargs)

```
frames = [Entity(parent=self, model=e.stem, enabled=False,
add_to_scene_entities=False) for e in model_names]
sequence = Sequence(loop=loop, auto_destroy=False)
is_playing = False
autoplay = autoplay
duration

start()
pause()
```

```
resume()
finish()
```

```
application.asset_folder = application.asset_folder.parent.parent / 'samples'

'''
Loads an obj sequence as a frame animation.
So if you have some frames named run_cycle_000.obj,
run_cycle_001.obj, run_cycle_002.obj and so on,
you can load it like this: FrameAnimation3d('run_cycle_')
'''

FrameAnimation3d('blob_animation_')
```

GridEditor(Entity)

ursina.prefabs.grid_editor

GridEditor(size=(32,32), palette=(' ', '#', '|', 'o'), **kwargs)

```
w, self.h = int(size[0]), int(size[1])
brush_size = 1
auto_render = True
cursor = Entity(parent=self, model=Quad(segments=0, mode='line'),
origin=(-.5,-.5), scale=(1/self.w, 1/self.h),
color=color.color(0,1,1,.5), z=-.1)
selected_char = palette[1]
palette = palette
prev_draw = None
start_pos = (0,0)
outline = Entity(parent=self, model=Quad(segments=0, mode='line',
thickness=1), color=color.cyan, z=.01, origin=(-.5,-.5))
undo_cache = list()
undo_index = 0
help_text = Text(
        text=dedent('''
            left mouse:    draw
            control(hold): draw lines
            alt(hold):     select character
            ctrl + z:      undo
            ctrl + y:      redo
        '''),
        position=window.top_left,
        scale=.75
        )
edit_mode = True

update()
draw(x, y)
input(key)
record_undo()
floodfill(matrix, x, y, first=True)
```

```
'''
pixel editor example, it's basically a drawing tool.
can be useful for level editors and such
here we create a new texture, but can also give it an existing
texture to modify.
'''
from PIL import Image
t = Texture(Image.new(mode='RGBA', size=(32,32), color=(0,0,0,1)))
from ursina.prefabs.grid_editor import PixelEditor
PixelEditor(texture=load_texture('brick'))

'''
same as the pixel editor, but with text.
'''
from ursina.prefabs.grid_editor import ASCIIEditor
ASCIIEditor()
```

PixelEditor(GridEditor)

ursina.prefabs.grid_editor

PixelEditor(texture, palette=(color.black, color.white, color.light_gray, color.gray, color.red, color.orange, color.yellow, color.lime, color.green, color.turquoise, color.cyan, color.azure, color.blue, color.violet, color.magenta, color.pink), **kwargs)

grid = [[texture.get_pixel(x,y) for y in range(texture.height)] for x in range(texture.width)]

render()
save()

```
'''
pixel editor example, it's basically a drawing tool.
can be useful for level editors and such
here we create a new texture, but can also give it an existing
texture to modify.
'''
from PIL import Image
t = Texture(Image.new(mode='RGBA', size=(32,32), color=(0,0,0,1)))
from ursina.prefabs.grid_editor import PixelEditor
PixelEditor(texture=load_texture('brick'))

'''
same as the pixel editor, but with text.
'''
from ursina.prefabs.grid_editor import ASCIIEditor
ASCIIEditor()
```

ASCIIEditor(GridEditor)

ursina.prefabs.grid_editor

ASCIIEditor(size=(61,28), palette=(' ', '#', '|', 'A', '/', '\\', 'o', '_', '-', 'i', 'M', '.'), font='VeraMono.ttf', color=color.black, line_height=1.1, **kwargs)

text_entity = Text(parent=self.parent, text=text, x=self.x, line_height=line_height, font=font)
scale = (self.text_entity.width, self.text_entity.height)

render()
input(key)

```
'''
pixel editor example, it's basically a drawing tool.
can be useful for level editors and such
here we create a new texture, but can also give it an existing
texture to modify.
'''
from PIL import Image
t = Texture(Image.new(mode='RGBA', size=(32,32), color=(0,0,0,1)))
from ursina.prefabs.grid_editor import PixelEditor
PixelEditor(texture=load_texture('brick'))

'''
same as the pixel editor, but with text.
'''
from ursina.prefabs.grid_editor import ASCIIEditor
ASCIIEditor()
```

HealthBar(Button)

ursina.prefabs.health_bar

HealthBar(max_value=100, show_text=True, show_lines=False, **kwargs)

bar = Entity(parent=self, model='quad', origin=self.origin, z=-.01, color=color.red.tint(-.2), ignore=True)
animation_duration = .1
lines = Entity(parent=self.bar, origin=self.origin, y=-1, color=color.black33, ignore=True)
roundness = .25
max_value = max_value
clamp = True
show_lines = show_lines

```
show_text = show_text
scale_x = self.scale_x # update rounded corners
scale_y = self.scale_y # update background's rounded corners
value = self.max_value
```

```
health_bar_1 = HealthBar(bar_color=color.lime.tint(-.25),
roundness=.5, value=50)

def input(key):
    if key == '+' or key == '+ hold':
        health_bar_1.value += 10
    if key == '-' or key == '- hold':
        health_bar_1.value -= 10
```

HotReloader(Entity)

ursina.prefabs.hot_reloader

HotReloader(path=__file__, **kwargs)

```
path = path
path = Path(self.path)
realtime_editing = False   # toggle with f8
hotkeys = {
        'ctrl+r' : self.reload_code,
        'f5'     : self.reload_code,
        'f6'     : self.reload_textures,
        'f7'     : self.reload_models,
        'f8'     : self.reload_shaders,
       # 'f9'     : self.toggle_hotreloading,
        }

input(key)
update()
toggle_hotreloading()
hot_reload()
reload_code(reset_camera=True)
reload_textures()
reload_models()
reload_shaders()
```

```
application.hot_reloader.path =
application.asset_folder.parent.parent / 'samples' / 'platformer.py'

'''
By default you can press F5 to reload the starting script, F6 to
reimport textures and F7 to reload models.
'''
```

ContentTypes

ursina.prefabs.input_field

ContentTypes(path=__file__, **kwargs)

```
background = Entity(model='quad', texture='pixelscape_combo',
parent=camera.ui, scale=(camera.aspect_ratio,1), color=color.white)
gradient = Entity(model='quad', texture='vertical_gradient',
parent=camera.ui, scale=(camera.aspect_ratio,1),
color=color.hsv(240,.6,.1,.75))

username_field = InputField(y=-.12, limit_content_to='0123456789')
password_field = InputField(y=-.18, hide_content=True)
username_field.next_field = password_field

def submit():
    print('ursername:', username_field.text)
    print('password:',  password_field.text)

Button('Login', scale=.1, color=color.cyan.tint(-.4), y=-.26,
on_click=submit).fit_to_text()
```

InputField(Button)

ursina.prefabs.input_field

InputField(default_value='', label='', max_lines=1, max_width=24, **kwargs)

```
default_value = default_value
limit_content_to = None
hide_content = False   # if set to True, will display content as '*'. can also be set to character instead of True.
next_field = None
text_field = TextField(world_parent=self, x=-.45, y=.3, z=-.1, max_lines=max_lines)
active = False
text
text_color

render()
input(key)
```

```
background = Entity(model='quad', texture='pixelscape_combo', parent=camera.ui, scale=(camera.aspect_ratio,1), color=color.white)
gradient = Entity(model='quad', texture='vertical_gradient', parent=camera.ui, scale=(camera.aspect_ratio,1), color=color.hsv(240,.6,.1,.75))

username_field = InputField(y=-.12, limit_content_to='0123456789')
password_field = InputField(y=-.18, hide_content=True)
username_field.next_field = password_field

def submit():
    print('ursername:', username_field.text)
    print('password:',  password_field.text)

Button('Login', scale=.1, color=color.cyan.tint(-.4), y=-.26, on_click=submit).fit_to_text()
```

MemoryCounter(Text)

ursina.prefabs.memory_counter

MemoryCounter(kwargs)**

```
parent = camera.ui
position = window.bottom_right - Vec2(.025,0)
origin = (0.5, -0.5)
process = psutil.Process(os.getpid())
i = 0
text = 'eofiwjeofiwejf'

update()
```

```
MemoryCounter()
'''
Displays the amount of memory used in the bottom right corner
'''
```

Panel(Entity)

ursina.prefabs.panel

Panel(kwargs)**

```
parent = camera.ui
model = Quad()
color = Button.color
```

```
p = Panel()
```

PlatformerController2d(Entity)

ursina.prefabs.platformer_controller_2d

PlatformerController2d(kwargs)**

```
model = 'cube'
origin_y = -.5
scale_y = 2
color = color.orange
collider = 'box'
animator = Animator({'idle' : None, 'walk' : None, 'jump' : None})
walk_speed = 8
walking = False
velocity = 0 # the walk diection is stored here. -1 for left and 1 for
right.
jump_height = 4
jump_duration = .5
jumping = False
max_jumps = 1
jumps_left = self.max_jumps
gravity = 1
grounded = True
air_time = 0   # this increase while we're falling and used when
calculating the distance we fall so we fall faster and faster instead
of linearly.
traverse_target = scene     # by default, it will collide with
everything except itself. you can change this to change the boxcast
traverse target.
gravity = 0

update()
input(key)
jump()
start_fall()
land()
```

```
camera.orthographic = True
camera.fov = 10

ground = Entity(model='cube', color=color.white33, origin_y=.5,
scale=(20, 10, 1), collider='box')
wall = Entity(model='cube', color=color.azure, origin=(-.5,.5),
scale=(5,10), x=10, y=.5, collider='box')
wall_2 = Entity(model='cube', color=color.white33, origin=(-.5,.5),
scale=(5,10), x=10, y=5, collider='box')
ceiling = Entity(model='cube', color=color.white33, origin_y=-.5,
scale=(1, 1, 1), y=1, collider='box')

def input(key):
    if key == 'c':
        wall.collision = not wall.collision
        print(wall.collision)


player_controller = PlatformerController2d(scale_y=2, jump_height=4,
x=3)
camera.add_script(SmoothFollow(target=player_controller, offset=
[0,1,-30], speed=4))

EditorCamera()
```

RadialMenu(Entity)

ursina.prefabs.radial_menu

RadialMenu(buttons=list(), **kwargs)

```
parent = camera.ui
buttons = buttons
open_at_cursor = True
open_duration = .25
bg = Panel(
        parent=self,
        model='quad',
        z=99,
        scale=999,
        collider='box',
        color=color.color(0,0,0,.1),
        enabled=False)
z = -99
scale = .075

on_enable()
input(key)
```

```
rm = RadialMenu(
    buttons = (
        RadialMenuButton(text='1'),
        RadialMenuButton(text='2'),
        RadialMenuButton(text='3'),
        RadialMenuButton(text='4'),
        RadialMenuButton(text='5', scale=.5),
        RadialMenuButton(text='6', color=color.red),
        ),
    enabled = False
    )
RadialMenuButton(text='6', color=color.red,x =-.5, scale=.06),
def enable_radial_menu():
    rm.enabled = True
cube = Button(parent=scene, model='cube', color=color.orange,
highlight_color=color.azure, on_click=enable_radial_menu)
EditorCamera()
```

RadialMenuButton(Button)

ursina.prefabs.radial_menu

RadialMenuButton(kwargs)**

```
rm = RadialMenu(
    buttons = (
        RadialMenuButton(text='1'),
        RadialMenuButton(text='2'),
        RadialMenuButton(text='3'),
        RadialMenuButton(text='4'),
        RadialMenuButton(text='5', scale=.5),
        RadialMenuButton(text='6', color=color.red),
        ),
    enabled = False
    )
RadialMenuButton(text='6', color=color.red,x =-.5, scale=.06),
def enable_radial_menu():
    rm.enabled = True
cube = Button(parent=scene, model='cube', color=color.orange,
highlight_color=color.azure, on_click=enable_radial_menu)
EditorCamera()
```

Sky(Entity)

ursina.prefabs.sky

Sky(kwargs)**

```
update()
```

Slider(Entity)

ursina.prefabs.slider

Slider(min=0, max=1, default=None, height=Text.size, text='', dynamic=False, **kwargs)

```
parent = camera.ui
vertical = False
min = min
```

```
max = max
default = default
step = 0
height = height
on_value_changed = None     # set this to a function you want to be
called when the slider changes
setattr = None              # set this to (object, 'attrname') to set
that value when the slider changes
label = Text(parent=self, origin=(0.5, 0), x=-0.025, text=text)
bg = Entity(parent=self, model=Quad(scale=(.525, height),
radius=Text.size/2, segments=3),
        origin_x=-0.25, collider='box', color=color.black66)
knob = Draggable(parent=self, min_x=0, max_x=.5, min_y=0, max_y=.5,
step=self.step,
        model=Quad(radius=Text.size/2, scale=(Text.size, height)),
collider='box', color=color.light_gray,
        text='0', text_origin=(0, -.55), z=-.1)
value = self.default
dynamic = dynamic    # if set to True, will call on_value_changed()
while dragging. if set to False, will only call on_value_changed()
after dragging.

bg_click()
drop()
update()
slide()
```

```
box = Entity(model='cube', origin_y=-.5, scale=1, color=color.orange)

def scale_box():
    box.scale_y = slider.value
    print(thin_slider.value)

slider = Slider(0, 20, default=10, height=Text.size*3, y=-.4, step=1,
on_value_changed=scale_box, vertical=True)

thin_slider = ThinSlider(text='height', dynamic=True,
on_value_changed=scale_box)

thin_slider.label.origin = (0,0)
thin_slider.label.position = (.25, -.1)
```

ThinSlider(Slider)

ursina.prefabs.slider

ThinSlider(*args, **kwargs)

```
box = Entity(model='cube', origin_y=-.5, scale=1, color=color.orange)

def scale_box():
    box.scale_y = slider.value
    print(thin_slider.value)

slider = Slider(0, 20, default=10, height=Text.size*3, y=-.4, step=1,
on_value_changed=scale_box, vertical=True)

thin_slider = ThinSlider(text='height', dynamic=True,
on_value_changed=scale_box)

thin_slider.label.origin = (0,0)
thin_slider.label.position = (.25, -.1)
```

Sprite(Entity)

ursina.prefabs.sprite

Sprite(texture=None, ppu=ppu, **kwargs)

```
model = 'quad'
texture = texture
ppu = ppu
aspect_ratio = self.texture.width / self.texture.height
```

```
scale_x = self.scale_y * self.aspect_ratio
```

```
camera.orthographic = True
camera.fov = 1
Sprite.ppu = 16
Texture.default_filtering = None
s = Sprite('brick', filtering=False)
```

TextField(Entity)

ursina.prefabs.text_field

TextField(kwargs)**

```
font = 'VeraMono.ttf'
line_height = 1
max_lines = 99999
text_entity = Text(
        parent
line_numbers = Text(
        parent
character_width = Text.get_width('a', font=self.font)
cursor_parent = Entity(parent=self, scale=(self.character_width,
-1*Text.size))
cursor = Entity(parent=self.cursor_parent, model='cube',
color=color.white33, origin=(-.5, -.5), scale=(.1, 1))
bg = Entity(parent=self.cursor_parent, model='cube',
color=color.dark_gray, origin=(-.5,-.5), z=1, scale=(120, 20),
collider='box', visible=False)
selection = None
selection_parent = Entity(parent=self.cursor_parent)
register_mouse_input = False
text = ''
replacements = dict()
on_undo = list()
on_redo = list()
shifted_keys = {
        '-' : '_',
        '.' : ':',
        ',' : ';',
        '\'' : '*',
        '<' : '>',
        '+' : '?',
        '0' : '=',
        '1' : '!',
        '2' : '"',
        '3' : '#',
        # '4' : '¤',
        '5' : '%',
        '6' : '&',
        '7' : '/',
        '8' : '(',
        '9' : ')',
alted_keys = {
        '\'' : '´',
        '0' : '}',
        '2' : '@',
        '3' : '£',
        '4' : '¤',
        '5' : '€',
        '7' : '{',
        '8' : '[',
        '9' : ']',
shortcuts = {
        'newline':          ('enter', 'enter hold'),
        'erase':            ('backspace', 'backspace hold'),
        'erase_word':       ('ctrl+backspace', 'ctrl+backspace hold'),
        'delete_line':      ('ctrl+shift+k',),
        'undo':             ('ctrl+z', 'ctrl+z hold'),
        'redo':             ('ctrl+y', 'ctrl+y hold', 'ctrl+shift+z',
'ctrl+shift+z hold'),
        # 'save':             ('ctrl+s',),
        # 'save_as':          ('ctrl+shift+s',),
        'indent':           ('tab',),
        'dedent':           ('shift+tab',),
        'move_line_down':   ('ctrl+down arrow', 'ctrl+down arrow
hold'),
        'move_line_up':     ('ctrl+up arrow', 'ctrl+up arrow hold'),
        # 'cut':              ('ctrl+x',),
        'copy':             ('ctrl+c',),
        'paste':            ('ctrl+v',),
        # 'select_all':       ('ctrl+a',),
        # 'toggle_comment':   ('ctrl+alt+c',),
        # 'find':             ('ctrl+f',),
```

```
blink_cursor()
add_text(s, move_cursor=True)
move_line(a, b)
erase()
delete_selected()
input(key)
render()
update()
select_all()
draw_selection()
```

```
window.x = 200

window.color = color.color(0, 0, .1)
Button.color = color._20
window.color = color._25

Text.default_font = 'consola.ttf'
Text.default_resolution = 16*2
te = TextField(max_lines=300, scale=1)
te.text = dedent('''
    Lorem ipsum dolor sit amet, consectetur adipiscing elit.
    Aliquam sapien tellus, venenatis sit amet ante et, malesuada
porta risus.
    Etiam et mi luctus, viverra urna at, maximus eros. Sed dictum
faucibus purus,
    nec rutrum ipsum condimentum in. Mauris iaculis arcu nec justo
rutrum euismod.
    Suspendisse dolor tortor, congue id erat sit amet, sollicitudin
facilisis velit.'''
    )[1:]
te.render()
```

Tilemap(GridEditor)

ursina.prefabs.tilemap

Tilemap(tilemap='', tileset='', tileset_size=(8,8), **kwargs)

```
grid = [[self.tilemap.get_pixel(x,y) for y in
range(self.tilemap.height)] for x in range(self.tilemap.width)]
tileset = tileset
tileset_size = tileset_size
model = Mesh()
texture = tileset
colliders = list()
auto_render = False
outline = Entity(parent=self, model=Quad(segments=0, mode='line',
thickness=1), color=color.cyan, z=.01, origin=(-.5,-.5),
enabled=self.edit_mode)
uv_dict = {
        '11111111' : [(4,1), (5,1), (6,1), (7,1)],      # fill
single_block_coordinates = [(4,0), (5,0), (6,0), (7,0)]
variation_chance = [0,0,0,0,1,1,1,2,2,3]
uv_margin = .002
```

```
update()
draw_temp(position)
input(key)
render()
save()
```

```
EditorCamera()
tilemap = Tilemap('tilemap_test_level', tileset='test_tileset',
tileset_size=(8,4), parent=scene)
camera.orthographic = True
camera.position = tilemap.tilemap.size / 2
camera.fov = tilemap.tilemap.height
```

```
Text('press tab to toggle edit mode', origin=(.5,0), position=
(-.55,.4))
```

Tooltip(Text)

ursina.prefabs.tooltip

Tooltip(text='', background_color=color.black66, **kwargs)

original_scale = self.scale

update()

```
app = Ursina()

    tooltip_test = Tooltip(
    '<scale:1.5><pink>' + 'Rainstorm' + '<scale:1> \n \n' +
'''Summon a <blue>rain
storm <default>to deal 5 <blue>water
damage <default>to <red>everyone, <default>including
<orange>yourself. <default>
Lasts for 4 rounds.'''.replace('\n', ' '),
        background_color=color.red
)

    tooltip_test.enabled = True
    app.run()
```

TrailRenderer(Entity)

ursina.prefabs.trail_renderer

TrailRenderer(thickness=10, color=color.white, end_color=color.clear, length=6, **kwargs)

renderer = Entity(
        model
update_step = .025

update()
on_destroy()

```
window.color = color.black
mouse.visible = False
player = Entity()
player.graphics = Entity(parent=player, scale=.1, model='circle')
trail_renderer = TrailRenderer(parent=player, thickness=100,
color=color.yellow, length=6)

pivot = Entity(parent=player)
trail_renderer = TrailRenderer(parent=pivot, x=.1, thickness=20,
color=color.orange)
trail_renderer = TrailRenderer(parent=pivot, y=1, thickness=20,
color=color.orange)
trail_renderer = TrailRenderer(parent=pivot, thickness=2,
color=color.orange, alpha=.5, position=(.4,.8))
trail_renderer = TrailRenderer(parent=pivot, thickness=2,
color=color.orange, alpha=.5, position=(-.5,.7))

def update():
    player.position = lerp(player.position, mouse.position*10,
time.dt*4)

    if pivot:
        pivot.rotation_z -= 3
        pivot.rotation_x -= 2

def input(key):
    if key == 'space':
        destroy(pivot)
```

SynthGUI(Entity)

ursina.prefabs.ursfx

light dark

Entity
Text
Button
mouse
raycaster

application
build
camera
color
curve
duplicate
input_handler
main
mesh_importer
scene
shader
string_utilities
text
texture_importer
ursinamath
ursinastuff
window

Audio
Collider
BoxCollider
SphereCollider
MeshCollider
CollisionZone
Color
CubicBezier
HitInfo
Keys
Light
DirectionalLight
PointLight
AmbientLight
SpotLight
Ursina
MeshModes
Mesh
Wait
Func
Sequence
Shader
Texture
Trigger
Empty
LoopingList
Vec2
Vec3

Animation
Animator
ButtonGroup
ButtonList
Node
Conversation
Cursor
DebugMenu
Draggable
DropdownMenuButton
DropdownMenu
EditorCamera
ExitButton
FileButton
FileBrowser
FileBrowserSave
FirstPersonController
FrameAnimation3d
GridEditor
PixelEditor
ASCIIEditor
HealthBar
HotReloader

SynthGUI(kwargs)**

wave_panel = Entity(parent=self, scale=.35, x=-0)
waveform = Entity(parent=self.wave_panel, scale_y=.75)
waveform_bg = Entity(parent=self.waveform, model='quad', origin=(-.5,-.5), z=.01, color=color.black66)
volume_slider = Slider(parent=self.wave_panel, x=-.05, vertical=True, scale=1.95, min=.05, max=1, default=.75, step=.01, on_value_changed=self.play)
wave_selector = ButtonGroup(('sine', 'triangle', 'square', 'noise'), parent=self.wave_panel, scale=.11, y=-.075)
pitch_slider = Slider(parent=self.wave_panel, y=-.25, scale=1.95, min=-36, max=36, default=0, step=1, on_value_changed=self.play, text='pitch')
pitch_change_slider = Slider(parent=self.wave_panel, y=-.325, scale=1.95, min=-12, max=12, default=0, step=1, on_value_changed=self.play, text='pitch change')
speed_slider = Slider(parent=self.wave_panel, scale=1.95, min=.5, max=4, default=1, step=.1, on_value_changed=self.play, y=-.4, text='speed')
coin_button = Button(text='coin', parent=self.wave_panel, scale=(.25, .125), origin=(-.5,.5), y=-.45, on_click=coin_sound)
knobs = [Draggable(parent=self.waveform, scale=.05, model='circle', color=color.light_gray, highlight_color=color.azure, position=default_positions[i], i=i, min_y=0, max_y=1) for i in range(5)]
line = Entity(parent=self.waveform, model=Mesh(vertices=[Vec3(0,0,0), Vec3(1,0,0)], mode='line', thickness=3), z=.01, color=color.light_gray)
bg = Entity(parent=self.wave_panel, model='wireframe_quad', origin=(-.5,-.5), z=.02, color=color.black, scale_x=1)
play_button = Button(text='>', parent=self.wave_panel, model='circle', scale=(.125, .125), color=color.azure, origin=(-.5,-.5), position=(-.075,1.025), on_click=self.play)
copy_button = Button(text='copy', parent=self.wave_panel, scale=(.25, .125), color=color.dark_gray, origin=(-.5,-.5), position=(-.075+.125+.025, 1.025), on_click=self.copy_code)
paste_button = Button(text='paste', parent=self.wave_panel, scale=(.25, .125), color=color.dark_gray, origin=(-.5,-.5), position=(self.copy_button.x + self.copy_button.scale_x + .025, 1.025), on_click=self.paste_code)
code_text = Text('', parent=self.wave_panel, scale=2, position=(self.paste_button.x+self.paste_button.scale_x+.025,1.11))
background_panel = Entity(model=Quad(radius=.025), parent=self.wave_panel, color=color.black66, z=1, origin=(-.5,-.5), scale=(1.125,1.75+.025), position=(-.1,-.6))
recipe

coin_sound()
drag(this_knob=knob)
drop(this_knob=knob)
update()
input(key)
copy_code()
paste_code(code="")
draw()
play()

```
app = Ursina()

gui = SynthGUI(enabled=False)

def toggle_gui_input(key):
    if key == 'f3':
        gui.enabled = not gui.enabled

Entity(input=toggle_gui_input)


if __name__ == '__main__':
    Sprite('shore', z=10, ppu=64, color=color.gray)
    gui.enabled = True
    app.run()
```

VideoRecorder(Entity)

ursina.prefabs.video_recorder

VideoRecorder(duration=5, name='untitled_video', **kwargs)

```
recording = False
file_path = Path(application.asset_folder) / 'video_temp'
i = 0
duration = duration
fps = 30
video_name = name
t = 0
max_frames = int(self.duration * self.fps)
frames = []

start_recording()
stop_recording()
update()
convert_to_gif()
```

```
window.size *= .5
from ursina.prefabs.first_person_controller import
FirstPersonController
from ursina.shaders import lit_with_shadows_shader
random.seed(0)
Entity.default_shader = lit_with_shadows_shader

ground = Entity(model='plane', collider='box', scale=64,
texture='grass', texture_scale=(4,4))

editor_camera = EditorCamera(enabled=False, ignore_paused=True)
player = FirstPersonController(model='cube', z=-10,
color=color.orange, origin_y=-.5, speed=8)
player.collider = BoxCollider(player, Vec3(0,1,0), Vec3(1,2,1))

gun = Entity(model='cube', parent=camera, position=(.5,-.25,.25),
scale=(.3,.2,1), origin_z=-.5, color=color.red, on_cooldown=False)

shootables_parent = Entity()
mouse.traverse_target = shootables_parent

for i in range(16):
    Entity(model='cube', origin_y=-.5, scale=2, texture='brick',
texture_scale=(1,2),
        x=random.uniform(-8,8),
        z=random.uniform(-8,8) + 8,
        collider='box',
        scale_y = random.uniform(2,3),
        color=color.hsv(0, 0, random.uniform(.9, 1))
        )


sun = DirectionalLight()
sun.look_at(Vec3(1,-1,-1))
Sky()

vr = VideoRecorder(duration=2)
def input(key):
    if key == '5':
        vr.start_recording()
    if key == '6':
        vr.stop_recording()
```

VideoRecorderUI(WindowPanel)

ursina.prefabs.video_recorder

VideoRecorderUI(kwargs)**

```
duration_label = Text('duration:')
duration_field = InputField(default_value='5')
fps_label = Text('fps:')
fps_field = InputField(default_value='30')
name_label = Text('name:')
name_field = InputField(default_value='untitled_video')
start_button = Button(text='Start Recording [Shift+F12]',
color=color.azure, on_click=self.start_recording)
```

y = .5
visible = False

input(key)
start_recording()

```
window.size *= .5
from ursina.prefabs.first_person_controller import
FirstPersonController
from ursina.shaders import lit_with_shadows_shader
random.seed(0)
Entity.default_shader = lit_with_shadows_shader

ground = Entity(model='plane', collider='box', scale=64,
texture='grass', texture_scale=(4,4))

editor_camera = EditorCamera(enabled=False, ignore_paused=True)
player = FirstPersonController(model='cube', z=-10,
color=color.orange, origin_y=-.5, speed=8)
player.collider = BoxCollider(player, Vec3(0,1,0), Vec3(1,2,1))

gun = Entity(model='cube', parent=camera, position=(.5,-.25,.25),
scale=(.3,.2,1), origin_z=-.5, color=color.red, on_cooldown=False)

shootables_parent = Entity()
mouse.traverse_target = shootables_parent

for i in range(16):
    Entity(model='cube', origin_y=-.5, scale=2, texture='brick',
texture_scale=(1,2),
        x=random.uniform(-8,8),
        z=random.uniform(-8,8) + 8,
        collider='box',
        scale_y = random.uniform(2,3),
        color=color.hsv(0, 0, random.uniform(.9, 1))
        )


sun = DirectionalLight()
sun.look_at(Vec3(1,-1,-1))
Sky()

vr = VideoRecorder(duration=2)
def input(key):
    if key == '5':
        vr.start_recording()
    if key == '6':
        vr.stop_recording()
```

Space()

ursina.prefabs.window_panel

Space(height=1)

height = height

```
'''
WindowPanel is an easy way to create UI. It will automatically layout
the content.
'''
wp = WindowPanel(
    title='Custom Window',
    content=(
        Text('Name:'),
        InputField(name='name_field'),
        Button(text='Submit', color=color.azure),
        Slider(),
        Slider(),
        ),
        popup=True,
        enabled=False
    )

def input(key):
    if key == 'space':
        wp.enabled = True
```

WindowPanel(Draggable)

ursina.prefabs.window_panel

WindowPanel(title='', content=[], **kwargs)

```
content = content
text = title
popup = False
panel = Entity(parent=self, model='quad', origin=(0,.5), z=.1,
color=self.color.tint(.1), collider='box')

layout()
on_enable()
close()
```

```
'''
WindowPanel is an easy way to create UI. It will automatically layout
the content.
'''
wp = WindowPanel(
    title='Custom Window',
    content=(
        Text('Name:'),
        InputField(name='name_field'),
        Button(text='Submit', color=color.azure),
        Slider(),
        Slider(),
        ),
        popup=True,
        enabled=False
    )

def input(key):
    if key == 'space':
        wp.enabled = True
```

chunk_mesh

ursina.scripts.chunk_mesh

```
app = Ursina()
t = time.time()
application.asset_folder = application.asset_folder.parent.parent
terrain = Entity(model=Terrain('grass_fields_heightmap', skip=8),
texture='grass', texture_scale=(3,3), scale=256)
grid = [[None for z in range(8)] for x in range(8)] # make 2d array of
entities
x_slices = 8
z_slices = 8
terrain.model.generated_vertices = [v+Vec3(.5,0.5) for v in
terrain.model.generated_vertices]
player = FirstPersonController(position=(0,200,0))
player.add_script(NoclipMode())
app.run()

update()
```

colorize

ursina.scripts.colorize

```
get_world_normals(model)
colorize(model, left=color.white, right=color.blue, down=color.red,
up=color.green, back=color.white, forward=color.white, smooth=True,
world_space=True, strength=1)
```

```
import random
for i in range(10):
    e = Entity(model=load_model('sphere',
path=application.internal_models_compressed_folder,
use_deepcopy=True))
    e.position =
(random.uniform(-3,3),random.uniform(-3,3),random.uniform(-3,3))
    e.rotation =
(random.uniform(0,360),random.uniform(0,360),random.uniform(0,360))
    e.scale = random.uniform(1,3)
    e.model.colorize(smooth=False, world_space=True, strength=.5)


Sky(color=color.gray)
EditorCamera()
```

combine

ursina.scripts.combine

temp_entity = None

combine(entity, analyze=False, auto_destroy=True, ignore=[])
get_vertices(entity, relative_to=None)

```
p = Entity()
e1 = Entity(parent=p, model='sphere', y=1.5, color=color.pink)
e2 = Entity(parent=p, model='cube', color=color.yellow, x=1,
origin_y=-.5)
e3 = Entity(parent=e2, model='cube', color=color.yellow, y=2,
scale=.5)

def input(key):
    if key == 'space':
        from time import perf_counter
        t = perf_counter()
        p.combine()
        print('combined in:', perf_counter() - t)



EditorCamera()
```

generate_normals

ursina.scripts.generate_normals

normalize_v3(arr)
generate_normals(vertices, triangles=None, smooth=True)

```
vertices = (
    (-0.0, -0.5, 0.0), (0.1, -0.48, -0.073), (-0.038, -0.48, -0.11),
    (0.361804, -0.22, -0.26), (0.3, -0.32, -0.22), (0.40, -0.25,
-0.14),
    (-0.0, -0.5, 0.0), (-0.038, -0.48, -0.11), (-0.03, -0.48, -0.11)
)
norms = generate_normals(vertices)
```

grid_layout

ursina.scripts.grid_layout

grid_layout(l, max_x=8, max_y=8, spacing=(0,0,0), origin=(-.5,.5,0),
offset=(0,0,0))

```
center = Entity(model='quad', scale=.1, color=color.red)
p = Entity()
for i in range(4*5):
    b = Button(parent=p, model='quad', scale=.5, scale_x=1,
text=str(i), color=color.tint(color.random_color(),-.6))
    b.text_entity.world_scale = 1
t = time.time()
grid_layout(p.children, max_x=7, max_y=10, origin=(0, .5))
```

```
center = Entity(parent=camera.ui, model=Circle(), scale=.005,
color=color.lime)
EditorCamera()
print(time.time() - t)
```

merge_vertices

ursina.scripts.merge_vertices

distance(a, b)
merge_overlapping_vertices(vertices, triangles=None, max_distance=.1)

```
verts = ((0,0,0), (1,0,0), (1,1,0), (0,0,0), (1,1,0), (0,1,0))
tris = (0,1,2,3,4,5)

new_verts, new_tris = merge_overlapping_vertices(verts, tris)
print('verts:', (verts), (new_verts))
print('tris:', (tris), (new_tris))


e = Entity(model=Mesh(new_verts, new_tris, mode='triangle'))
EditorCamera()
```

NoclipMode

ursina.scripts.noclip_mode

NoclipMode(speed=10, require_key='shift')

speed = speed
require_key = require_key
ignore_paused = True

input(key)
update()

```
player = Entity(model='cube', color=color.orange)
Entity(model='plane', scale=10)
EditorCamera()


player.add_script(NoclipMode2d())
```

NoclipMode2d

ursina.scripts.noclip_mode

NoclipMode2d(speed=10, require_key='shift')

speed = speed
require_key = require_key
ignore_paused = True

input(key)
update()

```
player = Entity(model='cube', color=color.orange)
Entity(model='plane', scale=10)
EditorCamera()


player.add_script(NoclipMode2d())
```

PositionLimiter()

ursina.scripts.position_limiter

PositionLimiter(min_x=-math.inf, max_x=math.inf, min_y=-math.inf, max_y=math.inf, min_z=-math.inf, max_z=math.inf)

```
min_x = min_x
max_x = max_x
min_y = min_y
max_y = max_y
min_z = min_z
max_z = max_z

update()
```

project_uvs

ursina.scripts.project_uvs

```
project_uvs(model, aspect_ratio=1, direction='forward',
regenerate=False)
```

```
e = Entity(model='sphere', texture='ursina_logo')
project_uvs(e.model)
EditorCamera()
```

Scrollable()

ursina.scripts.scrollable

Scrollable(kwargs)**

```
max = inf
min = -inf
scroll_speed = .05
scroll_smoothing = 16
axis = 'y'
target_value = None

update()
input(key)
```

```
'''
This will make target entity move up or down when you hover the
entity/its children
while scrolling the scroll wheel.
'''

p = Button(model='quad', scale=(.4, .8), collider='box')
for i in range(8):
    Button(parent=p , scale_y=.05, text=f'giopwjoigjwr{i}',
origin_y=.5, y=.5-(i*.05))

p.add_script(Scrollable())
```

SmoothFollow()

ursina.scripts.smooth_follow

SmoothFollow(target=None, offset=(0,0,0), speed=8, rotation_speed=0, rotation_offset=(0,0,0))

```
target = target
offset = offset
speed = speed
rotation_speed = rotation_speed
rotation_offset = rotation_offset

update()
```

```
player = Entity(model='cube', color=color.orange)

def update():
    player.x += held_keys['d'] * .1
    player.x -= held_keys['a'] * .1
```

```
e = Entity(model='cube')
sf = e.add_script(SmoothFollow(target=player, offset=(0,2,0)))

def input(key):
    global sf
    if key == '1' and sf in e.scripts:
        e.scripts.remove(sf)

EditorCamera()
```

terraincast

ursina.scripts.terraincast

prepare_terrain(terrain, debug=False, calculate_normals=True)
terraincast(origin,)

```
terrainEntity = Entity(model=Terrain('heightmap_1', skip=8),
                       scale=(20, 5, 20),
                       rotation=(30, 40, 50),
                       origin=(1, 1, 1),
                       texture='heightmap_1')

hit_entity = Entity(model='sphere', scale=0.1)
EditorCamera()


def update():
    hit = terraincast(camera.world_position, terrainEntity,
direction=camera.forward, debug=True)
    if hit:
        hit_entity.position = hit.world_point + hit.world_normal

        hit.entity.rotation_y += 2*time.dt

Sky()
```

models

'arrow'
'circle'
'cube'
'cube_uv_top'
'diamond'
'icosphere'
'line'
'plane'
'quad'
'scale_gizmo'
'sky_dome'
'sphere'
'wireframe_cube'
'wireframe_quad'

```
e = Entity(model='quad')
```

textures

'arrow_down'
'arrow_right'
'brick'
'circle'
'circle_outlined'
'cog'
'cursor'
'file_icon'
'folder'
'grass'
'heightmap_1'
'horizontal_gradient'

'noise'
'radial_gradient'
'rainbow'
'reflection_map_3'
'shore'
'sky_default'
'sky_sunset'
'test_tileset'
'tilemap_test_level'
'ursina_logo'
'ursina_wink_0000'
'ursina_wink_0001'
'vertical_gradient'
'vignette'
'white_cube'

```
e = Entity(model='cube', texture='brick')
```

shaders

basic_lighting_shader
colored_lights_shader
fresnel_shader
geom_shader
lit_with_shadows_shader
matcap_shader
normals_shader
projector_shader
texture_blend_shader
transition_shader
triplanar_shader
unlit_shader
camera_contrast
camera_empty
camera_grayscale
camera_outline_shader
camera_vertical_blur
fxaa
pixelation_shader
ssao

```
from ursina.shaders import normals_shader
e = Entity(shader=normals_shader)
```

Circle(Mesh)

ursina.models.procedural.circle

Circle(resolution=16, radius=.5, rotate=True, mode='ngon', **kwargs)

vertices = list()

```
e = Entity(model=Circle(8, mode='line', thickness=10),
color=color.color(60,1,1,.3))
print(e.model.recipe)
origin = Entity(model='quad', color=color.orange, scale=(.05, .05))
ed = EditorCamera(rotation_speed = 200, panning_speed=200)
```

Cone(Mesh)

ursina.models.procedural.cone

Cone(resolution=4, radius=.5, height=1, direction=(0,1,0), add_bottom=True, mode='triangle', **kwargs)

```
Entity(model=Cone(8, direction=(0,1,0)),
color=color.color(60,1,1,.3))
```

```
origin = Entity(model='quad', color=color.orange, scale=(.05, .05))
ed = EditorCamera(rotation_speed = 200, panning_speed=200)
```

Cylinder(Pipe)

ursina.models.procedural.cylinder

Cylinder(resolution=8, radius=.5, start=0, height=1, direction=(0,1,0), mode='triangle', **kwargs)

```
Entity(model=Cylinder(6, start=-.5), color=color.color(60,1,1,.3))
origin = Entity(model='quad', color=color.orange, scale=(5, .05))
ed = EditorCamera(rotation_speed = 200, panning_speed=200)
```

Grid(Mesh)

ursina.models.procedural.grid

Grid(width, height, mode='line', thickness=1, **kwargs)

```
width = width
height = height
```

```
Entity(model=Grid(2, 6))
```

Pipe(Mesh)

ursina.models.procedural.pipe

Pipe(base_shape=Quad, origin=(0,0), path=((0,0,0),(0,1,0)), thicknesses=((1,1),), look_at=True, cap_ends=True, mode='triangle', **kwargs)

```
base_shape = base_shape
```

```
path = (Vec3(0,0,0), Vec3(0,1,0), Vec3(0,3,0), Vec3(0,4,0),
Vec3(2,5,0))
thicknesses = ((1,1), (.5,.5), (.75,.75), (.5,.5), (1,1))
e = Entity(model=Pipe(path=path, thicknesses=thicknesses))
e.model.colorize()

EditorCamera()
origin = Entity(model='cube', color=color.magenta)
origin.scale *= .25
```

Plane(Mesh)

ursina.models.procedural.plane

Plane(subdivisions=(1,1), mode='triangle', **kwargs)

```
vertices, self.triangles = list(), list()
uvs = list()
```

```
front =  Entity(model=Plane(subdivisions=(3,6)), texture='brick',
rotation_x=-90)

_ed = EditorCamera()
Entity(model='cube', color=color.green, scale=.05)
```

QuadMesh(Mesh)

ursina.models.procedural.quad

QuadMesh(radius=.1, segments=8, aspect=1, scale=(1,1), mode='ngon', thickness=1)

```
vertices = [Vec3(0,0,0), Vec3(1,0,0), Vec3(1,1,0), Vec3(0,1,0)]
radius = radius
mode = mode
thickness = thickness
uvs = list()
vertices = [(v[0]-offset[0], v[1]-offset[1], v[2]-offset[2]) for v in self.vertices]
```

```
from time import perf_counter
t = perf_counter()
for i in range(100):
    Entity(model=Quad(scale=(3,1), thickness=3, segments=3, mode='line'), color = color.color(0,1,1,.7))
print('-------', (perf_counter() - t))

origin = Entity(model='quad', color=color.orange, scale=(.05, .05))
camera.z = -5
```

Terrain(Mesh)

ursina.models.procedural.terrain

Terrain(heightmap, skip=1, **kwargs)

```
heightmap = heightmap
skip = skip    # should be power of two.
width, self.depth = self.heightmap.width//skip, self.heightmap.height//skip
aspect_ratio = self.width / self.depth
height_values = asarray(img)
height_values = flip(self.height_values, axis=0)
height_values = swapaxes(self.height_values, 0, 1)
vertices, self.triangles = list(), list()
uvs = list()
normals = list()
height_values = [[j/255 for j in i] for i in self.height_values]
```

```
e = Entity(model=Terrain('heightmap_1', skip=16), scale=(20,5,20), texture='heightmap_1')
Entity(model='plane', scale=e.scale, color=color.red)
EditorCamera()
Sky()
```