End User Documentation

Introduction

This is the Pi Sense Engine Documentation. In this section, there are a list of classes and functions the engine has built in to make game development a breeze.

Game Structure

The structure of a game class can be seen in the code to the right. The game class must have the same name as the file name to be valid. It also must derive from the Game class, as that is where all the game related functions are stored, like the game loop.

```
# DemoGame.py
from SenseEngine import *

class DemoGame(Game):
    def __init__(self):
        Game.__init__(self)
        self.run()

if __name__ == "__main__":
    DemoGame()
```

When self.run() is called in the above code the game loop is started. The game loop does not take place in another thread! You must load the scene before you call run, because no code after run will execute, as run is the game loop.

GameObjects

```
gObj = GameObject("Name",
        Sprite.Load("Sprite"),
        [PhysicsObject()])
```

A GameObject is any object that is spawn-able into the game. It has a position vector, a list of components and most times, a Sprite.

The GameObject class takes some things on initialization, the object's name, it's sprite (can be None), and sometimes a list of components. A list of components is only needed if you want to add a new Component to the object on initialization.

Components

A component is any class that derives from the Component class. Classes like PhysicsObject and Transform are components. To check if a class is a component you can use the code below:

```
if issubclass(type(my_class), Component):
    print("Is a Component!")
```

This works because all components must derive from the Component class. This just checks if the type of my_class derives from Component.

Every Component has a variable, masterObject. It's equal to the object that Component is attached to.

You can also add Components to GameObjects after they're initialized, with GameObject.AddComponent(Component), like seen below.

```
gObj.AddComponent(PhysicsObject())
```

It is also posable to remove Components with GameObject.Remove(Component)

You can also use GetComponent(type) to get that type of component from that object. gObj.GetComponent(Transform)

will return the transform on gObj, if there is none, it will return None instead.

Transform

The position Component on all GameObjects. By default, all GameObjects have this added to them and there is no way to prevent that. This ensures that all objects have a position upon rendering and physics calculation.

To get/set the position, you can just GetComponent(Transform) on any GameObject.

```
print(
gObj.G
```

The above code would print the X position of gObj in the world.

PhysicsObject

A PhysicsObject is a Component meant for physics calculations. It holds the mass, velocity, decay (air resistance, friction, etc.). You don't have to specify any of these on initialization, but if you do, it may be beneficial to do something like this. Any value you don't state stays default.

```
foo =
Physic
```

Collider

The collider component takes in an array of Point classes in initialization. Each point will act as a point in space that can be collided with. The was to make the entire rendered object a collider is as follows:

Sprite

The Sprite class holds an array of points. Point.vector stores the position of that point, (0,0), being the objects position in the world. Point.color stores the color of that point in rgb.

```
foo = Point(Vector(0,0), Color(255,255,255))
```

SpriteRenderer

The SpriteRenderer class is what stores what an object looks like. SpriteRenderer.sprite is the sprite class that holds the actual points. On initialization SpriteRenderer takes an array of Points

Loading Sprites from Files

```
foo = SpriteRenderer([
    Point(Vector(0,0), Color(255,255,255)),
    Point(Vector(0,1), Color(255,255,255)),
    Point(Vector(1,0), Color(255,255,255)),
    Point(Vector(1,1), Color(255,255,255))
    ])
```

- You don’t have to manually punch in every value into code every time you want a new sprite however, you can load them in!

To the right is the format for a Sprite file. In brackets, there is the x, y coordinates of the point relative to the position. In Parentheses is the color of the point. To load a sprite just call Sprite.Load(location). Do not call this with an instance, you can just use it statically.

```
# fooSprite
[0,0] = (255,255,255)
[1.0] = (255,255,255)
[0.1] = (255,255,255)
[1,1] = (255,255,255)
```

When dealing with paths

```
foo = SpriteRenderer(
        Sprite.Load("[CWD]/fooSprite"))
```

to files, [CWD] is the current working directory of your game. This is 9/10 times the same place as your Game Python file.

Scripts

All the code you write often doesn't go in your Game Python file, but in custom scripts attached to objects.

```
# FooBehavior.py
from SenseEngine import *

class FooBehabior(Script):
    def __init__(self):
        Script.__init__(self)

    # Called every frame
    def Update(self):
        print("A Frame Passed!")
```

A script, like seen on the right, can be attached to an object just like a component can be. GameObject.AddComponent() works the same as it does for components as it does scripts. There are a few special callbacks you can use as well.

Script Callbacks

Update()

Update is called every frame.

Start()

Called on first run of script.

OnCollision(collider)

Called when one object collides with another. The argument collider is equal to the object that is being collided with.

Inputs

Input is handled by callback in Sense Engine. You call game.AddInputEvent(K_KEY, Function)

```
game.
AddInp
utEven
t(K_SP
ACE
```

The above code would print “Hello, World!” every time the space

key is pressed. On the next page, there is a short list of keycodes, the full list can found at:

https://www.pygame.org/docs/ref/key.html

GameObject Spawning

To actually put a GameObject into the game, you have to use a function called game.AddObject(). In this function you pass it a valid GameObject and it gets added to the game. There is also a DeleteObject, that removes GameObjects.

KeyASCII	ASCII	Common Name
K_0	0	0
K_1	1	1
K_2	2	2
K_3	3	3
K_4	4	4
K_5	5	5
K_6	6	6
K_7	7	7
K_8	8	8
K_9	9	9
K_a	a	a
K_b	b	b
K_c	c	c
K_d	d	d
K_e	e	e
K_f	f	f
K_g	g	g
K_h	h	h
K_i	i	i
K_j	j	j
K_k	k	k
K_l	l	l
K_m	m	m
K_n	n	n
K_o	o	o
K_p	p	p
K_q	q	q
K_r	r	r
K_s	s	s
K_t	t	t
K_u	u	u
K_v	v	v
K_w	w	w
K_x	x	x
K_y	y	y
K_z	z	z
K_UP		up arrow
K_DOWN		down arrow
K_RIGHT		right arrow
K_LEFT		left arrow
K_RSHIFT		right shift
K_LSHIFT		left shift
K_RCTRL		right ctrl
K_LCTRL		left ctrl
K_RALT		right alt