lutes Documentation

Release 1.0.0a1

Greizgh

April 06, 2015

Contents

Pronounced as the ancient French city “Lutèce”, Lutes is a micro component entity system engine.

CHAPTER 1

Component Entity System

Component entity system architecture is often used in game development.

It allows great flexibility by using composition over inheritence. It is based on three core elements:

Entity An entity is a mere ID that represent an object in our world

Component A component is a structure holding the object data for an aspect of the world. Several components can be associated to an entity.

System A system contain the logic for an aspect of the world

Lutes adds a fourth element: the manager. The manager is the glue that ties components, entities and systems together.

You can learn more on component entity system architecture on this wiki.

CHAPTER 2

What lutes gives you

In its current state, lutes gives you a simplistic structure with basic elements you can inherit from.

Lutes is an experiment but should be functionnal: please see how green are the badges below.

CHAPTER 3

Technical details

3.1 Lutes internals

3.1.1 Manager

class `lutes.`**`Manager`**

Manager handles entities and their components.

`add_component`(*entity*, *component*)

Add a component to an entity

`add_system`(*system*)

Add system to the world

`create_entity`()

Create an entity in the world and return its identifier.

Return type entity

`dispatch_event`(*event*, *data*)

Dispatch an event to all subscribers

Parameters

- **event** – event name as string
- **data** – data that will be passed to subscribers

`get_component`(*entity*, *component*)

Get an entity's component by its type Returns None if no component of given type was found

Return type component or None

`has_component`(*entity*, *component*)

Check that given entity has component

`init`()

Initialize systems

`remove_component`(*entity*, *component*)

Remove a component from an entity

`remove_entity`(*entity*)

Remove an entity from the world

`remove_system`(*system*)

Remove a system from the world

subscribe(*event*, *callback*)
Subscribe a callback to an event

Parameters

- **event** – event name as string
- **callback** – callable

update(*delta*)
Update every system

Parameters **delta** – time elpased since last update

3.1.2 System

class lutes.**System**(*priority=99*)
A system handles a set of components. It is responsible for updating them.

entities = None
Entities handled by the system

handled_components = None
Components the system needs to update entites

init()
Initialize the system

priority = None
System priority, lower is updated first

update(*delta*)
Update entities

Parameters **delta** – time elapsed since last update

3.1.3 Component

class lutes.**Component**(*entity=None*)
A component is a data bag attached to an entity

entity = None
Entity the component relates to

3.2 Errors

exception lutes.errors.**InvalidEntityError**(*entity*)
Exception raised when trying to process an invalid entity

exception lutes.errors.**LutesError**
Root lutes exception, use it to catch lutes related exceptions

l