Desarrollo rápido de videojuegos con el panda y la pitón

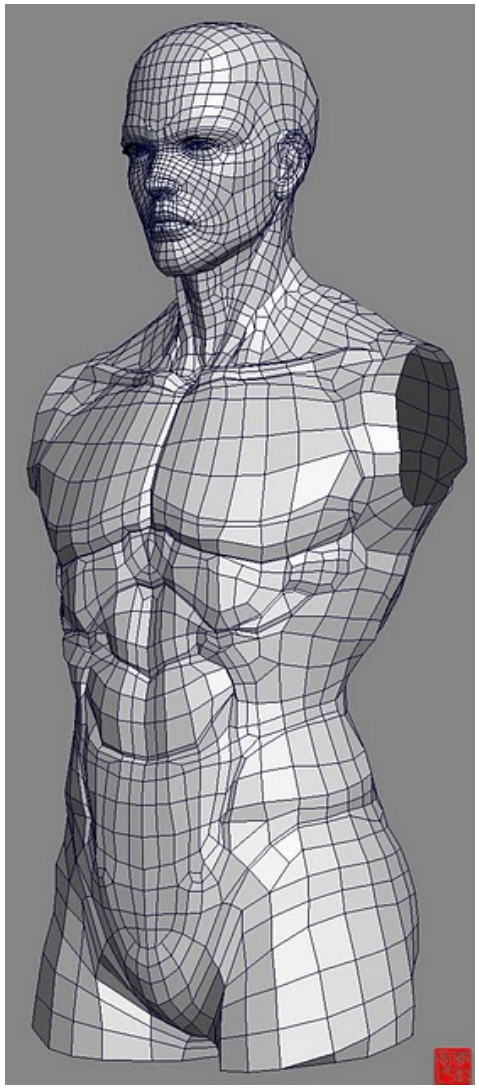
Conceptos aplicados con Python y Panda3D

Juan Pedro Bolívar Puente

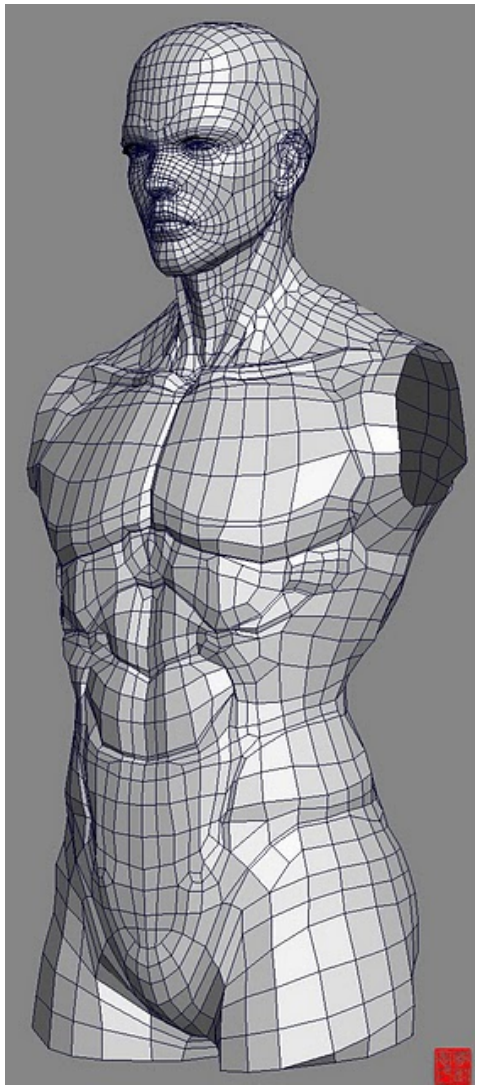
granada.hacklabs.org

Noviembre 2010

Índice



Índice



Desarrollo de videojuegos

Nuestro enfoque

Figura: Desarrollador de juegos
= Hombre orquesta del futuro

¡El desarrollo de videojuegos!

¡Disciplina compleja y amplia!

Inteligencia Artificial, Algoritmia, Geometría, Física, Redes, Hardware, Arte, Narrativa, ...

Nuestra aproximación

- Entender los **conceptos básicos** fundamentales.
- Entender los **principios arquitectónicos**.
- Usar **herramientas concretas** para programar **juegos**.

Nuestras herramientas (I)

Lenguaje de programación Python

Ventajas

- Tipado dinámico ⇒ Prototipado rápido.
- Orientación a objetos elegante ⇒ Buena arquitectura.
- Bueno para scripting ⇒ Modding.
- Soporte programación funcional ⇒ Expresividad.
- Software Libre.

Desventajas

Poco eficiente ⇒ Sólo para la lógica de alto nivel.

Nuestras herramientas (II)

Framework de videojuegos Panda3D

Ventajas

- Muy completo en funcionalidad.
- Corazón en C++ con bindings a Python.
- Software libre.
- Desarrollado por Disney y la Carniege Mellon University.

Desventajas

- Uso inapropiado de `__builtins__`.
- Posibles mejoras de diseño y *pitonización*.

Arquitectura Panda3d

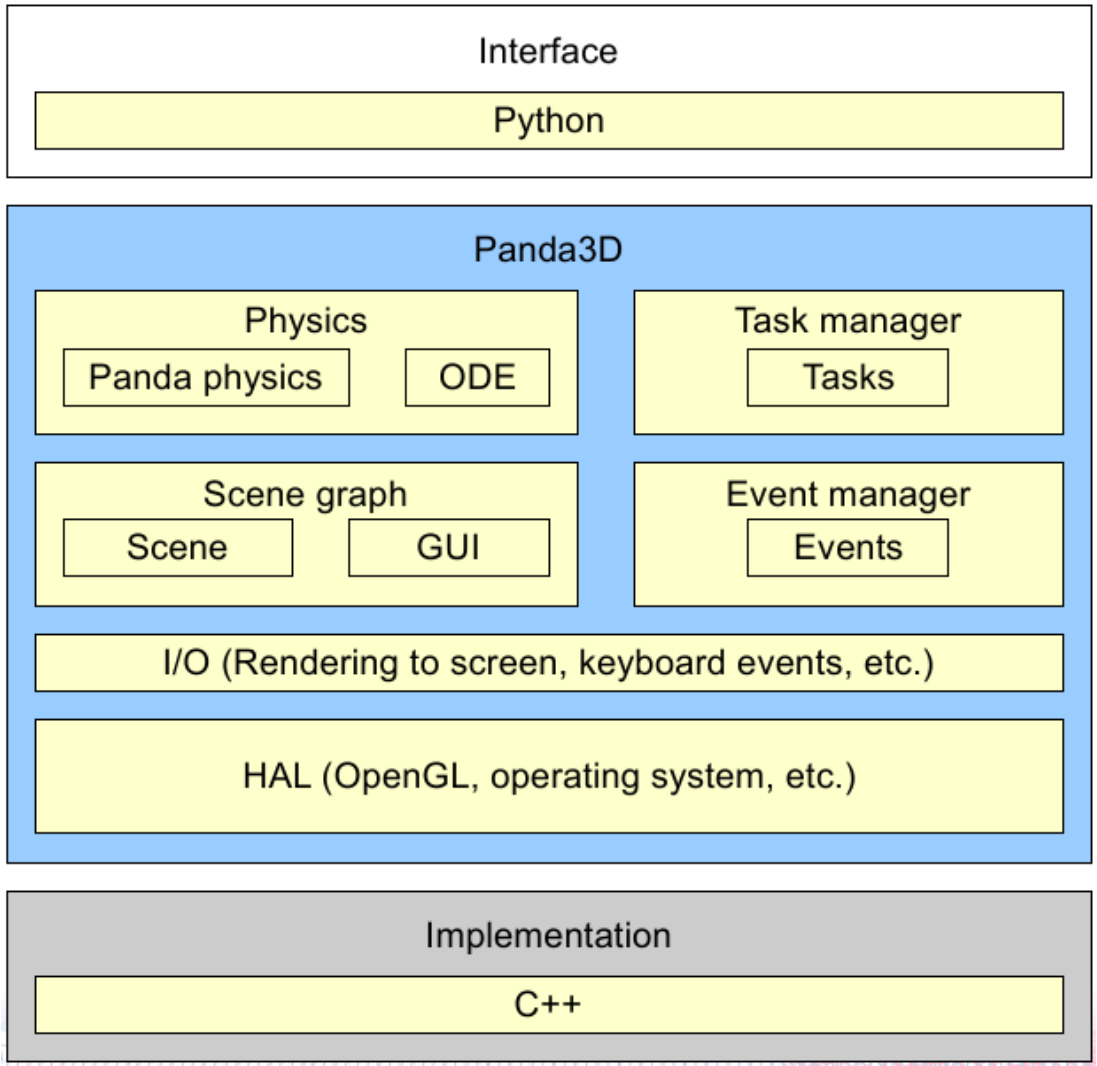
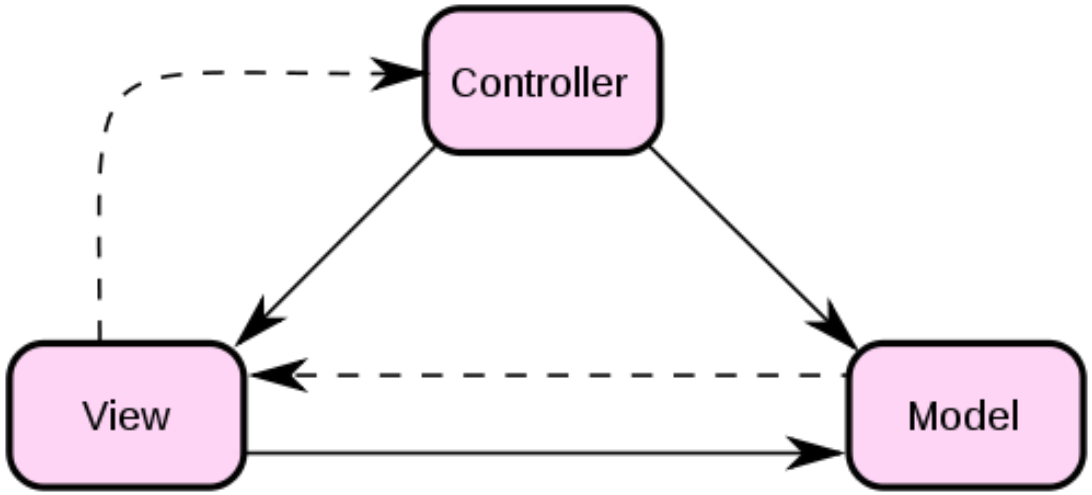


Figura: Arquitectura de Panda3d

Model-View-Controller con Panda3d



Modelo

- Datos estáticos del juego.
- Emiten eventos cuando cambian.

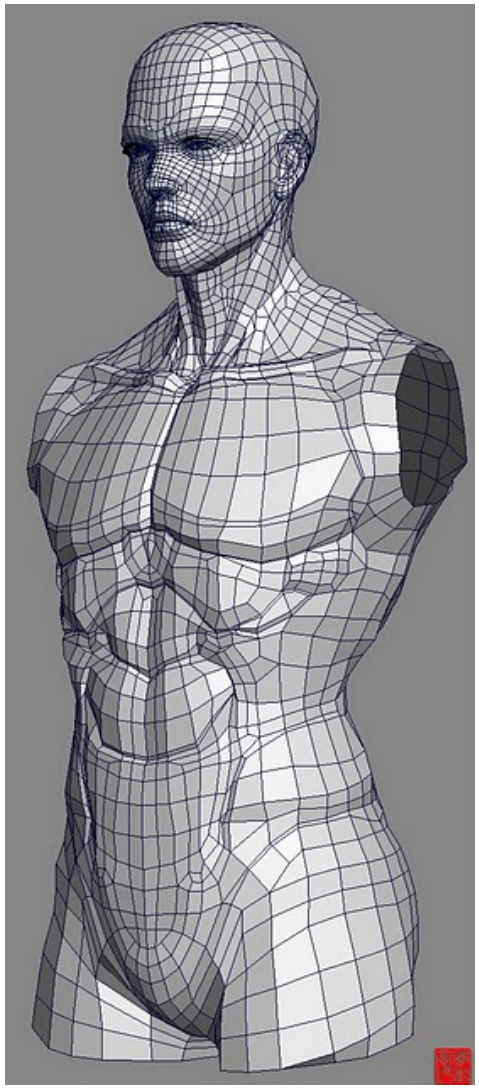
Vista

- Representación (*¿scenegraph?*).
- Reacciona a los eventos del modelo.

Controlador

- Lógica del juego.
- Tareas, eventos.
- Modifica modelo.

Índice



El bucle principal

Una vista de pájaro

Un juego es una simulación interactiva.

Un juego, en vista de pájaro

```
def juego ():
    inicializar_modelo ()
    while True:
        obtener_entrada_del_usuario ()
        actualizar_modelo ()
        actualizar_pantalla ()
```

El bucle principal

Abstraemos el gestor de tareas

Los problemas del modelo estático

```
# Si anadimos un nuevo tipo de entidad
class Pajarito (object):
    ...
# Tenemos que trastocar codigo lejano
def actualizar_modelo ():
    ...
    actualizar_pajaritos ()
def actualizar_pantalla ():
    ...
    pintar_pajaritos ()
```

El bucle principal

Abstraemos el bucle principal

El gestor de tareas

clase `TaskManager` (built-in `taskMgr`)

- `add (task, name, ...)`: Añade una tarea.
- `remove (task_or_name)` : Quita una tarea.
- `run ()` ó `step ()`: Ejecuta el bucle principal.

Las tareas

Funciones libres (*callables*) y clase `Task`

- Controlan su ejecución devolviendo:
  - `Task.cont` Se ejecuta de nuevo en el siguiente frame.
  - `Task.done` o `None`: No se ejecuta más.
  - `Task.again`: Se ejecutará de nuevo después de `delayTime`

Abstraemos el bucle principal

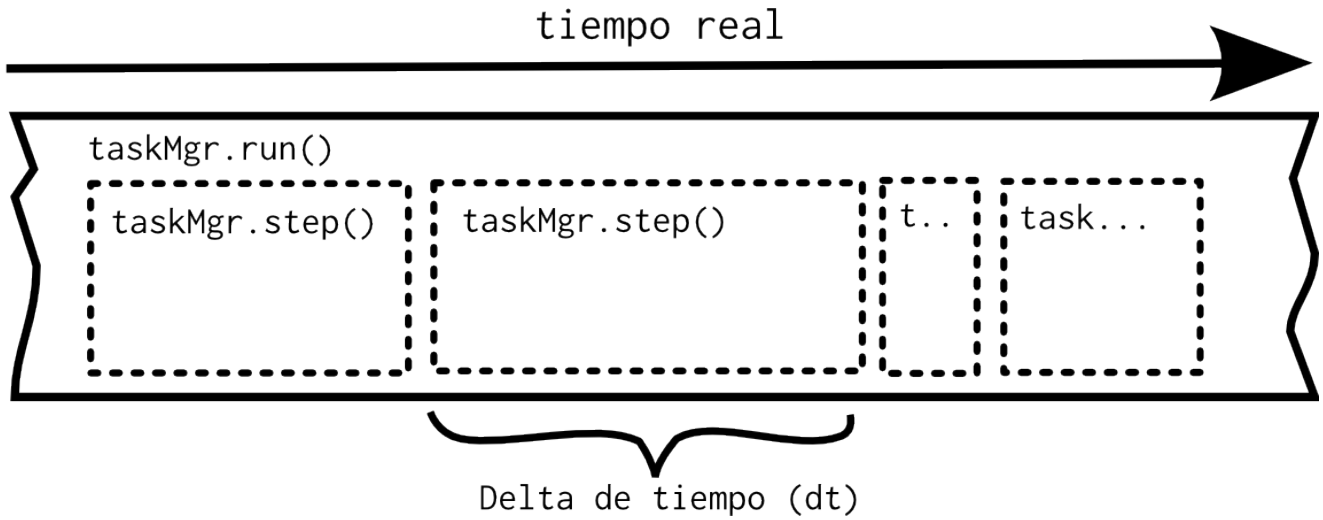
Ejemplo de código

code0.py

```
import direct.directbase.DirectStart

def hola_mundo (task):
    print "Hola mundo!"
    return task.cont

taskMgr.add (hola_mundo, "hola-mundo")

run ()
```

Gestion del tiempo

¡El tiempo de simulación es variable!




```
class Coche (object):
    def actualizar_posicion (self, task):
        self.posicion += dt * self.velocidad
        # Cuanto vale 'dt' ?!?!?!
```

Gestion del tiempo

¡El tiempo de simulación es variable!

Clase `ClockObject` (builtin `globalClock`)

- `getDt ()`: Obtiene el delta de tiempo
- `setMaxDt (maxdt)`: Establece el limite delta.
- Lleva cuenta de numero de iteraciones, FPS medios y otras cosas.
- Panda3d añade automáticamente una tarea que lo actualiza.
- También podemos calcularlo a partir de `task.time`.

```
self.posicion += globalClock.getDt () * self.velocidad
```

Animaciones basadas en tareas

Los *intervalos* de Panda3D

Intervalo = Tarea que se ejecuta durante un periodo de tiempo.

- Tiene unos métodos `start ()`, `stop ()`, `loop ()`, ...
- Se puede reproducir una subparte `start (start, end, rate)`
- Existen con diferentes usos, pueden combinarse:
  - Intervalos de control `Sequence` y `Parallel`.
  - Ejecutar funciones `Func` y `LerpFunc`.
  - Esperar como `Wait`
  - Mover modelos, reproducir sonidos, animaciones, etc...

Animaciones basadas en tareas

Ejemplo de código

code1.py

```
import direct.directbase.DirectStart
from direct.interval.IntervalGlobal import *
def print_sth (something):
    print something
intervalo = Sequence (
    Parallel (LerpFunc (print_sth, fromData=0,
                        toData=10, duration=1),
              LerpFunc (print_sth, fromData=10,
                        toData=0, duration=1)),
    Wait (1.0),
    Func (print_sth, "Hola Mundo"))
intervalo.start ()
run ()
```

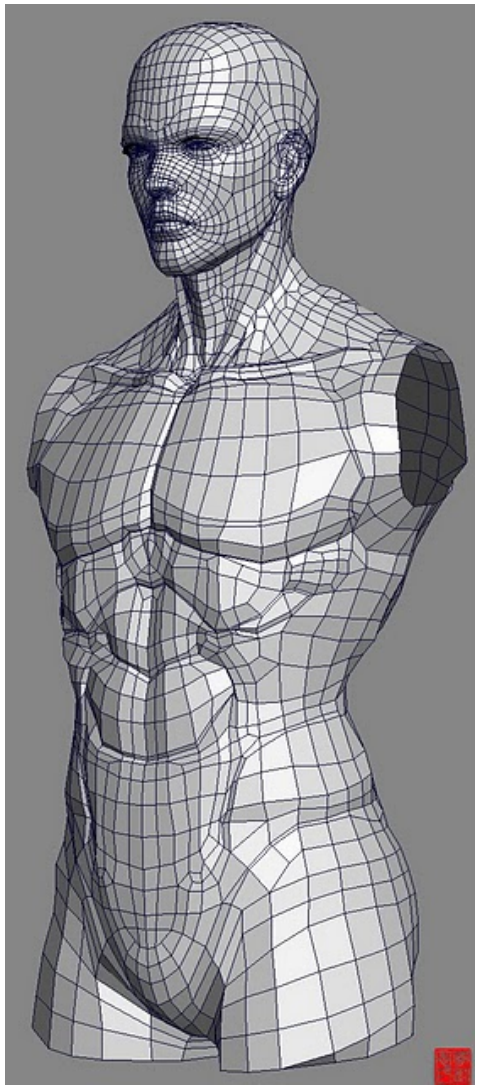
Animaciones basadas en tareas

Ejecución del código

Salida code1.py

```
0.0
10.0
0.599999986589
9.40000001341
0.630000010133
9.36999998987
...
9.6899998188
0.310000181198
9.86000001431
0.139999985695
10
0
Hola Mundo
```

Índice



Mostrando cosas por pantalla

¡Panda3D se encarga de todo!

Figura: Editando un modelo en Blender

Trabajamos con **objetos** en un mundo 3D (`PandaNode`)

- Modelos 3D (importados de Blender) (`ModelNode`)
- Elementos geométricos (`GeomNode`)
- Luces (`LightNode`)
- Cámara (`CameraNode`)

Cada objeto tiene **propiedades**

- Posición (`setPos/getPos`)
- Rotación (`setHpr/getHpr`)
- Escalado (`setScale/getScale`)

El grafo de escena

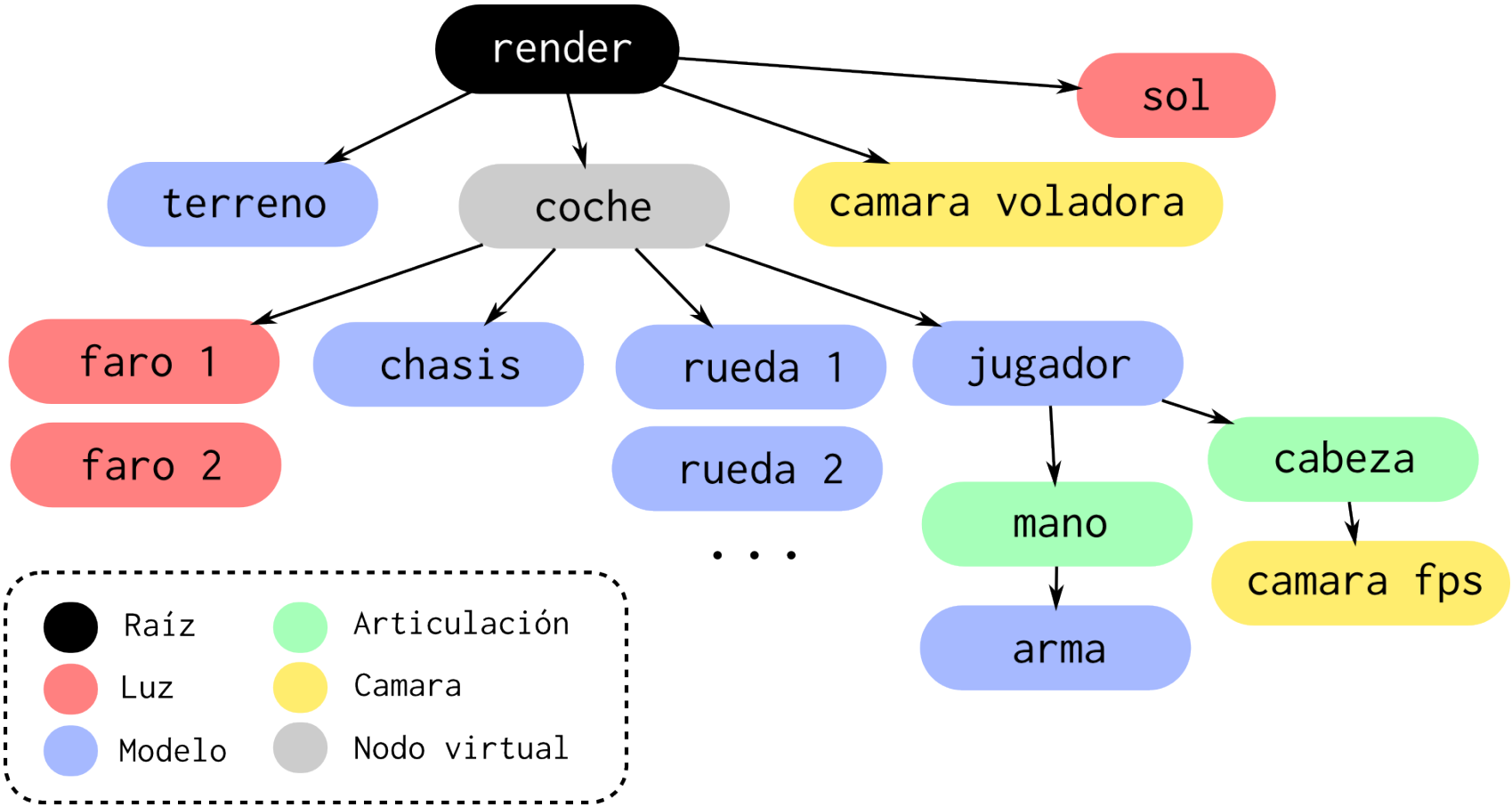


Figura: Las propiedades son **relativas** al padre

Manipulando el grafo de escena

Un **NodePath** es un "puntero" (*handle* a un nodo del grafo).

- `reparentTo (node)`: Cambia el padre del nodo.
- `wrtReparentTo (node)`: Cambia el padre del nodo conservando las propiedades.
- `removeNode ()`: Desconecta el nodo del grafo.
- `attachNewNode (name_or_node)`: Asocia un nodo y lo devuelve con un path.
- `instanceTo (node)`: Asocia el nodo a otro nodo.
  - *Instancing*, salir de la estructura arbórea.
  - "Clonar" nodos eficientemente.

Cargando recursos (*assets*)

La clase `Loader` (built-in **`loader`**) sirve para cargar recursos.

- `loadModel (name, ...)`: Carga un modelo.
- `loadSfx (name, ...)`: Carga un sonido.
- `loadMusic (name, ...)`: Carga un sonido con *buffering*.
- `loadTexture (name, ...)`: Carga una imagen.
- `loadFont (name, ...)`: Carga una fuente.
- ¡Usa una caché!
- `unload[resource] (res)`: Borra un recurso de la caché.

Un ejemplo más elaborado

Diseñamos el grafo de escena.

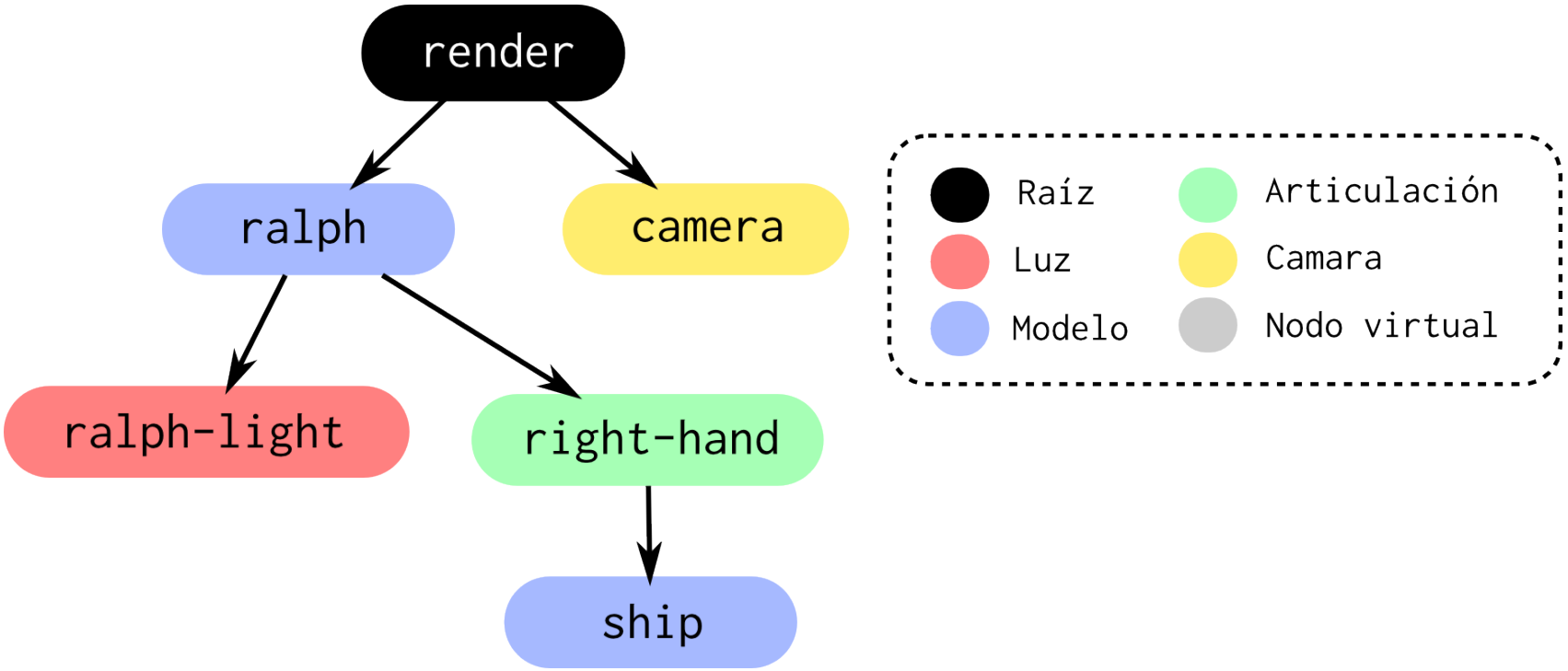


Figura: Grafo de escena de nuestro ejemplo.

Un ejemplo más elaborado

code2.py (I)

```
import direct.directbase.DirectStart
from direct.actor.Actor import Actor
from pandac.PandaModules import DirectionalLight, VBase4
from functools import partial
import math
base.disableMouse ()

# Forma alternativa de cargar un modelo con animaciones,
# cargamos el modelo de un nino con cara de cabron.
ralph_node = Actor ('models/ralph.egg.pz',
                    { 'run' : 'models/ralph-run.egg.pz'})
ralph_node.reparentTo (render)
ralph_node.setBlend (frameBlend = True)

# Le regalamos un juguete al nino y lo ponemos en su mano.
ship_node = loader.loadModel ('models/ship.bam')
```

Un ejemplo más elaborado

Niños que corren, luces, juguetes...

code2.py (II)

```
ship_node.reparentTo (ralph_node.exposeJoint (
    None, 'modelRoot', 'RightHand'))
ship_node.setScale (.2)

# Ponemos una luz encima del nino.
light_node = ralph_node.attachNewNode (
    DirectionalLight ('ralph-light'))
light_node.node ().setColor (VBase4 (1., .5, .5, 1.))
light_node.setPos (0, 0, 10)
light_node.lookAt (0, 0, 0)
light_node.reparentTo (ralph_node)
render.setLight (light_node)
```

Un ejemplo más elaborado

...y animaciones!

code2.py (III)

```
def circles_task (node, task):
    node.setPos (math.sin (task.time) * 10.0,
                 math.cos (task.time) * 10.0, 0)
    node.setHpr (- (task.time * 180 / math.pi - 90), 0, 0)
    return task.cont

camera.setPos (30, 0, 30)
camera.lookAt (0, 0, 0)

taskMgr.add (partial (circles_task, ralph_node), 'circles')
ralph_node.loop ('run')
run ()
```

Un ejemplo más elaborado

¡Ejecutamos!

Figura: Ejecutando `code2.py`

Pintando la interfaz de usuario

El grafo de escena para 2D

`render2d` es la raíz de un grafo de escena renderizado con proyección ortogonal en primer plano para la interfaz.

- **`aspect2d`** es un hijo de `render2d` escalado tal que la zona visible es:

$$\begin{cases} x \in [-ratio, ratio] \\ y \in [-1, 1] \end{cases} \tag{1}$$

Dónde $ratio = \frac{alto\_ventana}{ancho\_ventana}$

Pintando la interfaz de usuario

La familia de clases `DirectGUI`

Las clases de **`DirectGUI`**

- Son un `NodePath`
- Se registran automáticamente en `aspect2d`.

- `DirectLabel` Una etiqueta de texto.
- `DirectButton` Un botón pulsable.
- `DirectEntry` Una entrada de texto.
- ...

¡No olvidar llamar a **`destroy ()`**!

Pintando la interfaz de usuario

Propiedades de `DirectGUI`

Casi todas las propiedades se pueden configurar en el constructor.

text	El texto del elemento.
image	Nombre de fichero o textura para la imagen del elemento.
pos	Tupla (x, y, z) con la posición del elemento.
text_fg	Tupla (r, g, b, a) con el color del texto.
text_bg	Tupla (r, g, b, a) con el color del fondo.
geom	`NodePath` con un modelo 3D para el elemento.
command	Función a llamar cuando se pulsa el elemento.
...	

Añadimos interfaz a nuestro ejemplo

code3.py

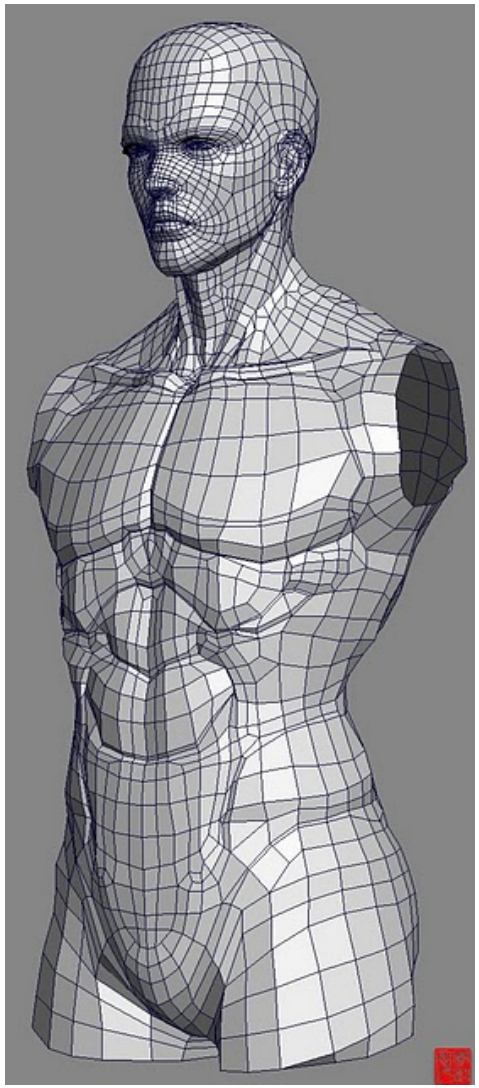
```
from direct.gui.DirectGui import *

...

texto = OnscreenText (
    text       = "Bienvenido␣al␣juego␣de␣Ralf␣dando␣vueltas.",
    scale      = .07,
    mayChange = True)
DirectButton (
    text       = "Botoncillo",
    pos        = (0, 0, -.15),
    scale      = .1,
    command    = lambda: texto.setText ("Boton␣pulsado!"))

...
```

Añadimos interfaz a nuestro ejemplo

¡Ejecutamos!

Figura: Ejecutando `code3.py`

Índice



Desacoplando el diseño

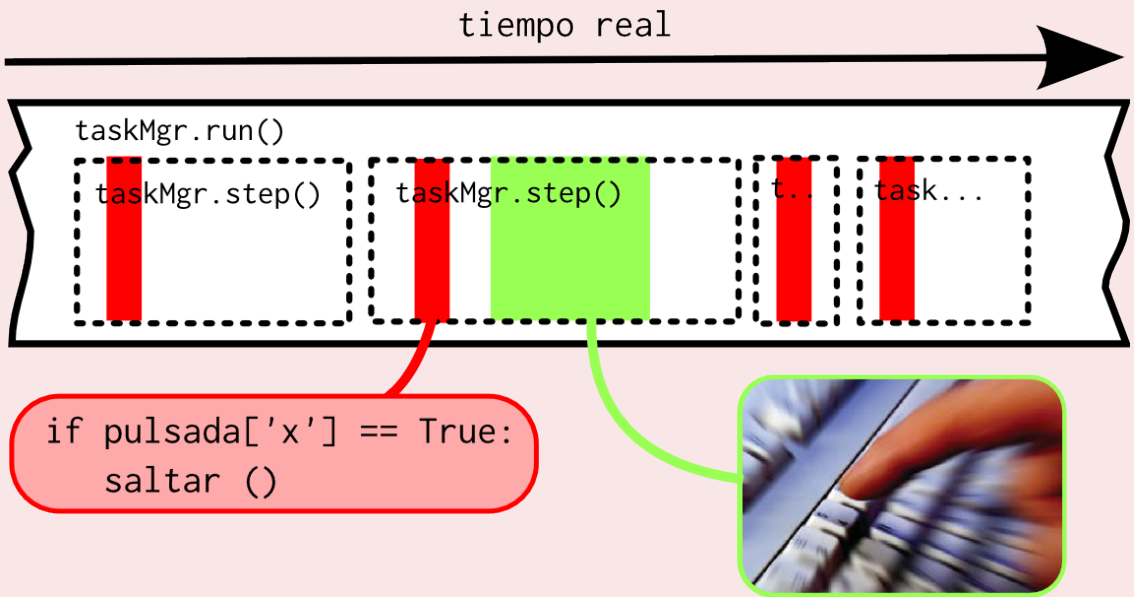
Problemas

1. ¿Como desacoplar el modelo de las vistas y los controladores?

   Ejemplo: Añadir soporte juego en red.

2. ¿Como añadir entrada de teclado y ratón?

   Espera activa no funciona.

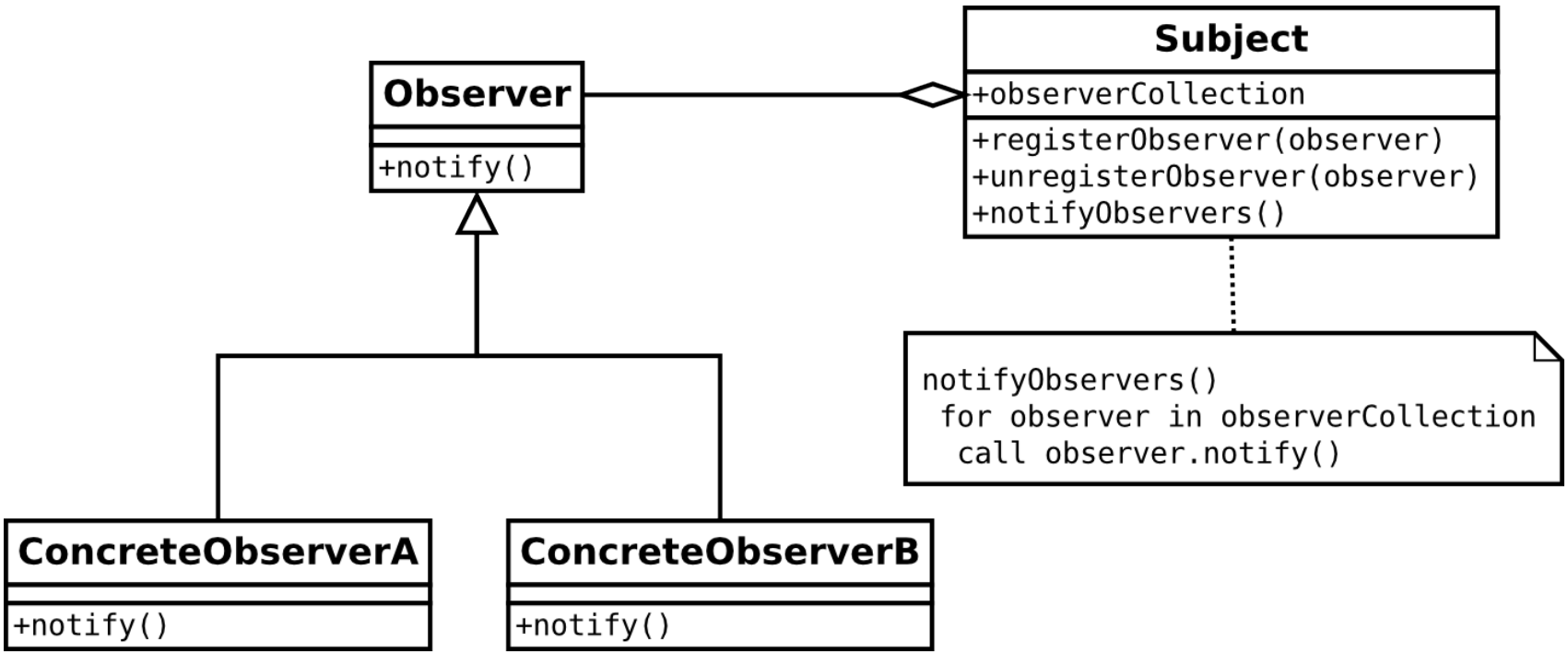


Desacoplando el diseño

El patrón *Observador*

Solución

El *patrón de diseño* observador.



El patrón *observador* en Panda3D

El sujeto

Centralizado en la clase `Messeger` (built-in `messenger`)

- `send (evento, ...)` Envía un evento con los argumentos dados.

El observador

Heredamos de la clase `DirectObject`[a]

- `accept (evento, func)` Registrarse en `evento` con `func`.
- `ignore (evento)` Desregistrarse de `evento`.
- `ignoreAll ()` Desregistrarse de todo.

[a] También sirve para tener tareas locales a un objeto.

Entrada y salida mediante `Messenger`

Panda3D emite automáticamente eventos de entrada y salida.

Eventos de teclado

a	o b, c, ...cualquier letra en minúsculas o nombre especial (arrow_left, enter, etc.). Se dispara al pulsar esa tecla.
a-up	Se dispara al levantar la tecla.
a-repeat	Se dispara al pulsar un rato la tecla.
shift-a	Se dispara con combinaciones de teclas. También se puede usar control o alt y combinarse.

Eventos de raton

mouse1	O 2, 3, botón del ratón.
mouse1-up	Botón del ratón levantado.
wheel_up	O down, al mover la rueda del ratón.

¡No hay mouse-move!

code4.py (I)

```
def make_mouse_move_task ():
    mouse = base.mouseWatcherNode
    old   = [ None, None ]
    if mouse.hasMouse ():
        old = [ mouse.getMouseX (), mouse.getMouseY () ]
    def mouse_move_task (task):
        if mouse.hasMouse ():
            new = [ mouse.getMouseX (), mouse.getMouseY () ]
            if new != old:
                if not (None in old):
                    messenger.send ('mouse-move',
                                    [(new [0] - old [0],
                                      new [1] - old [1])])
                old [0], old [1] = new
        return task.cont
    return mouse_move_task
```

Probando nuestra nueva tarea...

code4.py (II)

```
import direct.directbase.DirectStart

...

# Tampoco hay nada para 'aceptar' funciones libres
def accept_fn (event, function, *a, **k):
    obj = DirectObject ()
    obj.function = function
    obj.accept (event, obj.function, *a, **k)
    return obj

def print_sth (something):
    print something

accept_fn ('mouse-move', print_sth)
taskMgr.add (make_mouse_move_task (), 'mouse-move')
run ()
```

¡Salta, Ralph, salta!

Rediseñando el grafo de escena.

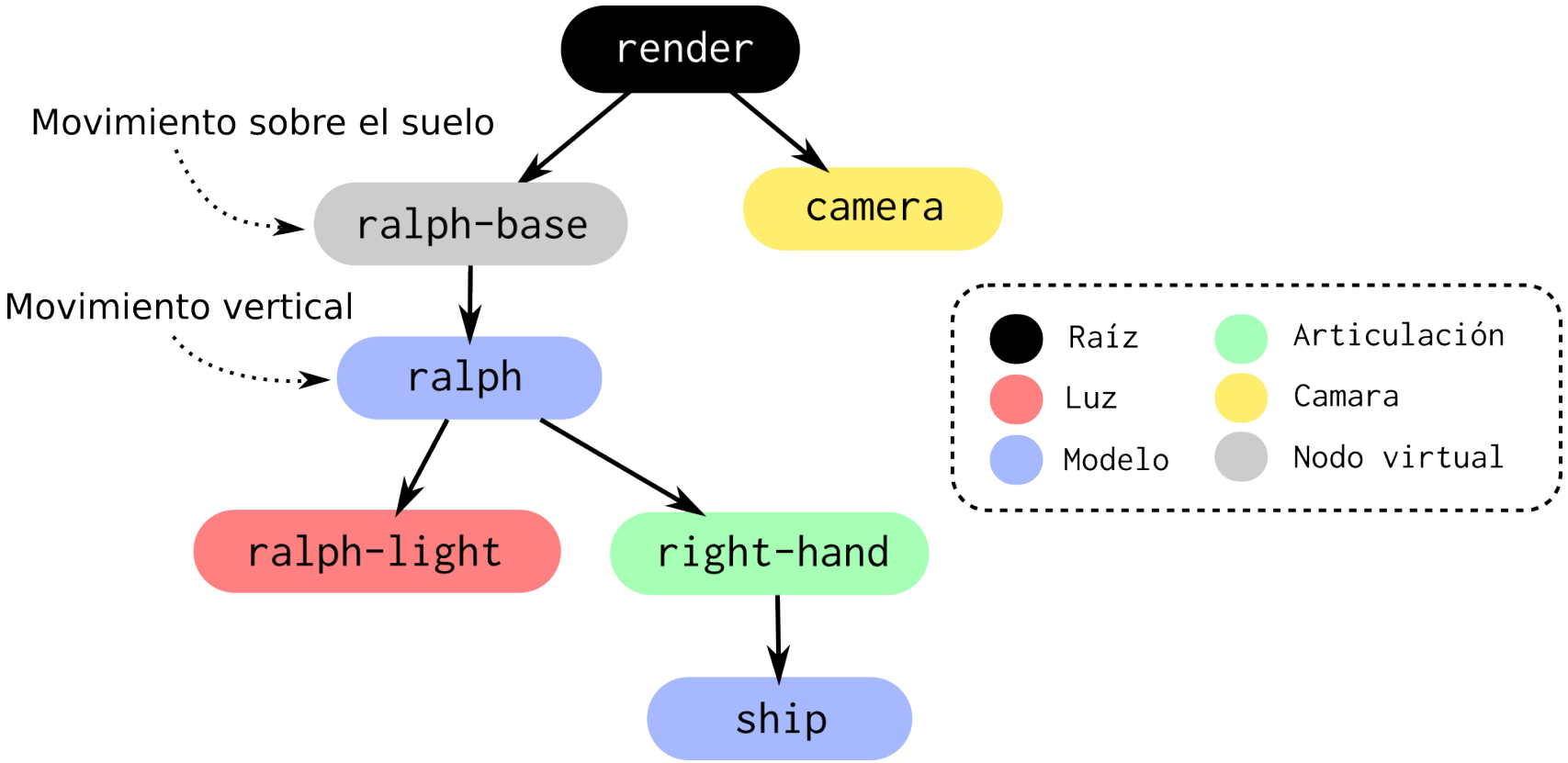


Figura: Grafo de escena preparado para que Ralph salte

¡Salta Ralph, salta!

Creamos una clase para Ralph...

code5.py (I)

```
class Ralph (DirectObject, object):
    def __init__ (self, *a, **k):
        super (Ralph, self).__init__ (*a, **k)
        # Esta es la base que permanece a nivel del suelo
        base_node = render.attachNewNode ("ralph-base")
        ralph_node = Actor (
            'models/ralph.egg.pz',
            { 'run'  : 'models/ralph-run.egg.pz' })
        ralph_node.reparentTo (base_node)
        ralph_node.setBlend (frameBlend = True)
        self.base_node  = base_node
        self.ralph_node = ralph_node
        # Continua con el resto del grafo de escena ...
```

¡Salta Ralph, salta!

...definimos la lógica del salto en base a eventos...

code5.py (II)

```
        self.jump_sfx      = loader.loadSfx ('sfx/jump.ogg')
        self.jump_done_sfx = loader.loadSfx ('sfx/fall.ogg')
        self.jump_anim     = Sequence (
            ralph_node.posInterval (.5, Vec3 (0, 0, 5)),
            ralph_node.posInterval (.5, Vec3 (0, 0, 0)))
        self.jump_anim.setDoneEvent ('ralph-jump-done')
        self.accept ('ralph-jump',      self.on_jump)
        self.accept ('ralph-jump-done', self.on_jump_done)

    def on_jump (self):
        if not self.jump_anim.isPlaying ():
            self.jump_sfx.play ()
            self.jump_anim.start ()
    def on_jump_done (self):
        self.jump_done_sfx.play ()
```

¡Salta Ralph, salta!

...y conectamos los eventos.

code5.py (III)

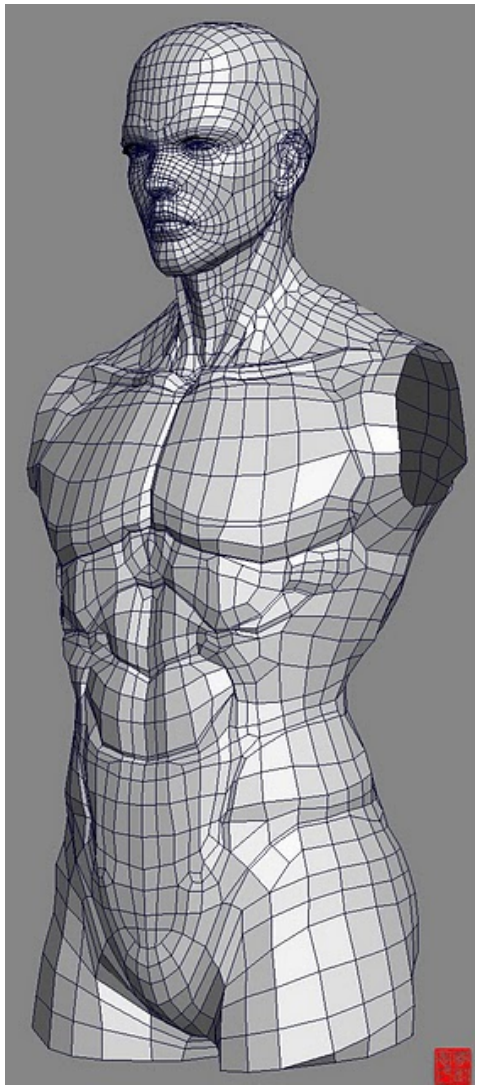
```
...

DirectButton (
    text       = "Salta!",
    pos        = (0, 0, -.15),
    scale      = .1,
    command    = lambda: messenger.send ('ralph-jump'))
# Forwadeamos la tecla espacio a ralph-jump
accept_fn ('space', lambda: messenger.send ('ralph-jump'))
ralph = Ralph ()
ralph.ralph_node.loop ('run')
taskMgr.add (partial (circles_task, ralph.base_node), 'c')
run ()
```

¡Salta, Ralph, salta!

¡Ejecutamos!

Figura: Ejecutando `code5.py`

Índice



Conclusión

... uf, qué alivio, ¡ya se acaba!

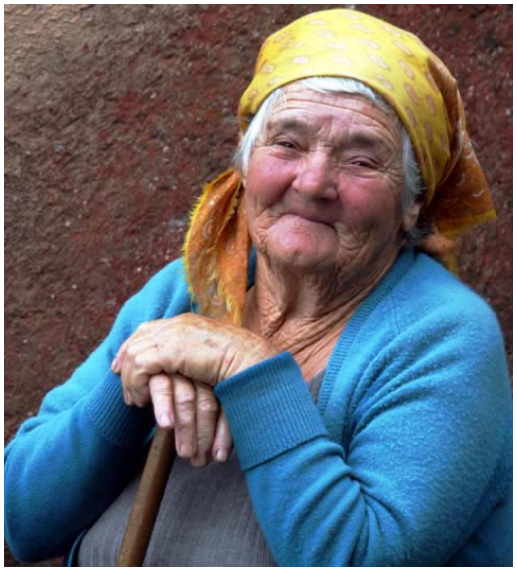


Figura: Haced caso a esta señora

La moraleja de la vieja ...

- Abstrae el bucle principal mediante tareas.
- Usa intervalos controlar sucesos en el tiempo.
- Usa el patrón observador para desacoplar.
- La *espera activa* es de niñas.

Y en el próximo capítulo...

- Detección de colisiones.
- Motores de físicas.
- *Shaders* y efectos especiales.
- Otros motores: Ogre3D, etc.

Recursos adicionales

Panda3D

Carnegie Mellon University y Disney

http://www.panda3d.org

Python

Guido van Rossum

http://www.python.org

Pigeoncide

Juan Pedro Bolívar Puente, Alberto Villegas Erce, Marc Modrow, Sari Sariola

https://savannah.nongnu.org/projects/pigeoncide/

Game Coding Complete.

Mike McShaffry.

Paraglyph Publishing, 2003.

Recursos adicionales

Overdose
Juan Pedro Bolívar Puente
http://suicidesoft.com/raskolnikov/overdose-0.1.1.tar.gz

Game Architecture and Design
Andrew Rollings, Dave Morris
New Riders Publishing, 2004

Algorithms and Networking for Computer Games
Jouni Smed and Harri Hakonen
John Wiley & Sons, 2006

Design Patterns. Elements of Reusable Object-Oriented Software.
Erich Gamma, Richard Helm, Ralph E. Johnson, and John Vlissides.
Addison-Wesley, March 1995.

Recursos adicionales

GameDev

`http://www.gamedev.net/`

Amit's Game Programming Information

`http://www-cs-students.stanford.edu/~amitp/gameprog.html`

¿Preguntas?

Muchas gracias por su atención.