glucosa Documentation

Release 0.1

Hugo Ruscitti / Enrique Porta

December 11, 2013

Contents

1 Introducción **3**
- 1.1 Motivación . . . 3
- 1.2 Instalación . . . 3
- 1.3 ¿Cómo empezar? . . . 3

2 Tutorial de programación básico **5**
- 2.1 Comenzando . . . 5
- 2.2 MainLoop . . . 6
- 2.3 Colocando al primer personaje . . . 7
- 2.4 Eventos . . . 8
- 2.5 Arreglo menor sobre el sprite . . . 9
- 2.6 El método `on_update` . . . 9
- 2.7 Muchos, muchos, muchos sprites! . . . 9
- 2.8 Finalizando... ¡No olvides ver la API! . . . 10

3 Cómo usar glucosa desde Sugar XO **11**

4 Creando una actividad para Sugar XO **15**
- 4.1 Introducción . . . 15
- 4.2 Generando la actividad . . . 16
- 4.3 Instalación . . . 16
- 4.4 Cambiando el ícono de la actividad . . . 16
- 4.5 Actualizando la actividad . . . 18
- 4.6 ¿Dónde se instalan las actividades? . . . 18
- 4.7 Distribuyendo la actividad . . . 19

5 API **21**

6 Índices y tablas **23**

Contenido:

CHAPTER 1

Introducción

Glucosa es una biblioteca super pequeña, persada para facilitar el desarrollo de videojuegos y programas multimedia.

1.1 Motivación

Existen muchas bibliotecas para hacer videojuegos, incluso la mayoría son excelentes bibliotecas, completas y atractivas. Entonces, ¿Por qué hacer una nueva biblioteca?.

Glucosa es una biblioteca impulsada por dos necesidades, por una lado: queríamos tener una biblioteca pequeña y fácil de utilizar para proyectos sencillos. Y por otro lado, los creadores de glucosa también estamos trabajando para llevar pilas-engine al sistema **sugar xo**, y necesitábamos algo de código base para comenzar.

Así que nos preguntamos, ¿por qué no?.

1.2 Instalación

Existen varias formas de instalar glucosa en tu sistema. Aquí veremos dos formas de instalar la biblioteca:

Una opción es clonar el repositorio **git** completo:

```
git clone git@github.com:hugoruscitti/glucosa.git
```

Dentro del directorio que se generará, encontrarás ejemplos y el archivo mas importante para que todo funcione: *glucosa.py*.

Otra opción, sobretodo cuando estás usando la biblioteca dentro de otro proyecto, es descargar solamente el archivo `glucosa.py` para incluir en tu proyecto:

```
wget https://raw.github.com/hugoruscitti/glucosa/master/glucosa.py
```

1.3 ¿Cómo empezar?

El repositorio de glucosa tiene varios ejemplos dentro del directorio `examples`, te recomendamos comenzar por ahí, viendo qué hace cada ejemplo, ejecutándolos para darte una idea de lo que glucosa puede hacer.

El siguiente paso es revisar esta documentación, incluso tenemos un *Tutorial de programación básico* para comenzar. Aquí hemos incluido mucha documentación sobre cómo funciona la biblioteca y todos los elementos que la componen.

CHAPTER 2

Tutorial de programación básico

En este documento veremos cómo construir un programa con glucosa paso a paso. Utilizaremos funciones de la biblioteca básica, así que las instrucciones son válidas tanto para programar desde GNU/Linux cómo dentro de Sugar XO.

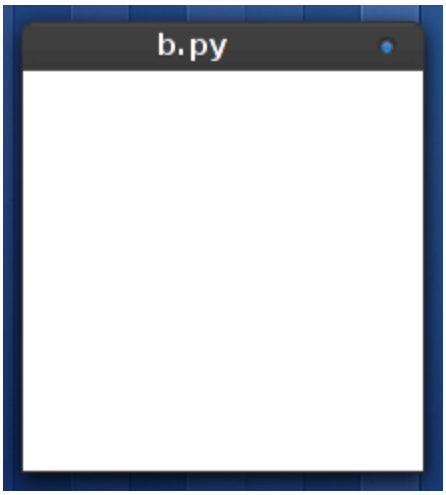
2.1 Comenzando

Glucosa funciona gracias a un sistema de eventos provisto por gtk, así que tu código tiene escribirse siguiendo una mínima estructura que le permita funcionar junto a gtk sin problemas.

Internamente, el area de dibujo de `glucosa` es un widget de `gtk`, así que si conoces `gtk` esto va a ser sencillo.

Comencemos la estructura de un programa estilo `glucosa`:

```
# -*- coding: utf-8 -*-
import pygtk
import gtk
import glucosa

class Game:

    def __init__(self):
        (self.window, self.canvas) = glucosa.create_window()
        self.mainloop = glucosa.MainLoop(self, self.canvas, fps=60)

    def on_update(self):
        pass

    def on_draw(self, context):
        pass

if __name__ == '__main__':
    juego = Game()
    gtk.main()
```



La primer parte del código realmente importante es `juego = Game()` y `gtk.main()`. Estas dos lineas inician el juego y lo mantienen en funcionamiento (respectivamente).

La clase `Game` se utiliza para representar al programa completo, el método `__init__` construye una ventana, un widget para dibujar (canvas) e inicializa un objeto `MainLoop`.

Veamos un poco mas de cerca a `MainLoop`:

2.2 MainLoop

Los videojuegos son un poco especiales, si bien son programas cómo cualquier otro, funcionan de forma ligeramente diferente.

En los programas convencionales, el funcionamiento está gobernado por el usuario. El programa queda esperando a que el usuario haga algo, y cuando el usuario lo hace el programa responde.

En un videojuego es diferente, el videojuego tiene que hacer algo todo el tiempo, los personajes tienen que caminar, las naves seguir volando etc.

Aquí es donde entra en juego `MainLoop`. Este objeto se encarga de mantener al juego activo, haciendo cosas todo el tiempo, independientemente de que el usuario inicie alguna acción. El objeto `MainLoop` espera ser iniciado con al menos dos parámetros:

```
glucosa.MainLoop(controller, canvas, fps)
```

- `controller`: es el objeto que se encargará de actualizar y redibujar la ventana, suele ser el mismo objeto que inicia el MainLoop. Es importante que el objeto que coloquemos ahí tenga los métodos `on_update` y `on_draw`.
- `canvas`: es un objeto `gtk.DrawingArea`, se puede crear con la función `glucosa.create_window` o directamente usando `gtk` por nuestra cuenta.
- `fps`: es un número entero, que indicará la velocidad de ejecución del juego. Es el acrónimo en inglés de "Frames Per Second", e indica cuantas veces se llamará a la función "on_update" por segundo. Poner esto a 60 es lo ideal, aunque algunos juegos usan 30 o incluso 24...

2.3 Colocando al primer personaje

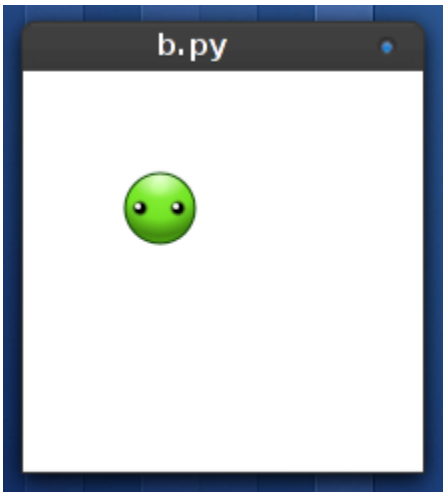
Los personajes de glucosa se llaman `sprites`, y consisten en objetos que representan un punto en la pantalla y tienen asociada una imagen.

Por ejemplo, para crear un `Sprite` podríamos escribir:

```
imagen = glucosa.Image('data/aceituna.png')
mi_sprite = glucosa.Sprite(imagen)
```

Una vez creado el personaje, es importante tener una referencia a él y también dibujarlo en cada actualización. Ampliemos un poco el código inicial:

```
# -*- coding: utf-8 -*-
import pygtk
import gtk
import glucosa

class Game:

    def __init__(self):
        (self.window, self.canvas) = glucosa.create_window()
        self.mainloop = glucosa.MainLoop(self, self.canvas, fps=60)

        imagen = glucosa.Image('data/aceituna.png')
        self.mi_sprite = glucosa.Sprite(imagen, 50, 50)

    def on_update(self):
        pass

    def on_draw(self, context):
        self.mi_sprite.draw(context)

if __name__ == '__main__':
    juego = Game()
    gtk.main()
```



Entonces, en pantalla tendríamos que ver al personaje es la esquina superior de la pantalla.

Para alterar al personaje se pueden usar muchos de sus atributos, un personaje tiene una posición, una rotación, una escala etc...

Por ejemplo, podríamos mover el personaje un poco hacia abajo, que gire unos grados y aparezca del doble de su tamaño cambiando con el siguiente código:

```
self.mi_sprite.x = 50
self.mi_sprite.y = 30
self.mi_sprite.scale = 2
self.mi_sprite.rotation = 30
```

2.4 Eventos

Hasta ahora nuestro personaje no es interactivo, solamente aparece en la pantalla y permanece dibujado sin moverse.

Para que el personaje interactúe, tendríamos que implementar el manejo de eventos.

Los eventos en glucosa son administrados por la clase `Events`, y se pueden conectar directamente a funciones para conocer el momento exacto de cada interacción del usuario.

Agreguemos algo de código para que el personaje de nuestra prueba persiga al puntero del `mouse`:

```
# -*- coding: utf-8 -*-
import pygtk
import gtk
import glucosa

class Game:

    def __init__(self):
        (self.window, self.canvas) = glucosa.create_window()
        self.mainloop = glucosa.MainLoop(self, self.canvas, fps=60)

        imagen = glucosa.Image('data/aceituna.png')
        self.mi_sprite = glucosa.Sprite(imagen)

        self.events = glucosa.Events(self.canvas)
        self.events.on_mouse_move += self.move_sprite

    def move_sprite(self, event):
        self.mi_sprite.x = event['x']
        self.mi_sprite.y = event['y']

    def on_update(self):
        pass

    def on_draw(self, context):
        self.mi_sprite.draw(context)

if __name__ == '__main__':
    juego = Game()
    gtk.main()
```

Es decir, el administrador de eventos (`glucosa.Events`), se inicializa una vez, y expone varias funciones asociadas a eventos para que le podamos conectar cosas.

Cada vez que el usuario mueva el mouse, el administrador de eventos detectará el movimiento y llamará a nuestro método `move_sprite`. Y en nuestro código, analizamos las coordenadas del mouse y las utilizamos para cituar al personaje en pantalla.

2.5 Arreglo menor sobre el sprite

Habrás notado que el personaje se sitúa un poco a la derecha del cursor, los sprites en glucosa tienen algo que se llama centro de rotación o punto de control, e inicialmente está en la posición 0 (la esquina superior izquierda del sprite).

Esto se puede corregir fácilmente cambiando los valores de los atributos `anchor_x` y `anchor_y`:

```
self.my_sprite.anchor_x = 18
self.my_sprite.anchor_y = 18
```

2.6 El método `on_update`

Cuando creamos el objeto `MainLoop`, creamos una clase llamada `Game` con un método llamado `on_update`.

Este método es muy útil, porque nos permitirá mantener en actualización a los personajes de un juego, e incluso controlar colisiones, temporizadores u objetivos.

En nuestro caso, para observar el funcionamiento general de glucosa, vamos a cambiar el código del método `on_update` para lograr que el personaje dé vueltas sobre sí mismo:

```
class Game:
    [...]

    def on_update(self):
        self.my_sprite.rotation += 1
```

Con este cambio, el personaje comenzará a dar vueltas, a una velocidad de 60 grados por segundo, porque de hecho la función `on_update` se ejecuta a esa frecuencia (`fps=60`).

Es una buena idea dejar el valor `fps` en un valor fijo, y regular la velocidad mediante lógica de nuestro juego. Por ejemplo, si queremos que el personaje dé vueltas mas rápido podríamos escribir:

```
self.my_sprite.rotation += 5
```

2.7 Muchos, muchos, muchos sprites!

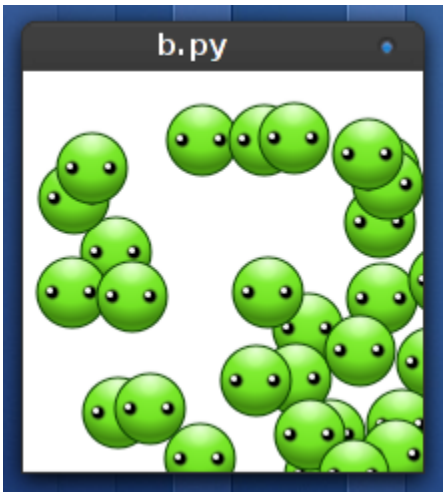
Glucosa no tiene límites de personajes, se pueden crear tanto cómo queramos.

Así que para simplificar el manejo de personajes, se suele crear una lista y almacenar a todos los personajes ahí.

El siguiente código genera 30 personajes:

```
# -*- coding: utf-8 -*-
import pygtk
import gtk
import glucosa
import random

class Game:

    def __init__(self):
        (self.window, self.canvas) = glucosa.create_window()
        self.mainloop = glucosa.MainLoop(self, self.canvas, fps=60)

        self.sprites = []
```

```
        for x in range(30):
            self.crear_un_personaje()

    def crear_un_personaje(self):
        imagen = glucosa.Image('data/aceituna.png')
        x = random.randint(0, 200)
        y = random.randint(0, 200)
        mi_sprite = glucosa.Sprite(imagen, x, y)
        self.sprites.append(mi_sprite)

    def on_update(self):
        for s in self.sprites:
            s.update()

    def on_draw(self, context):
        for s in self.sprites:
            s.draw(context)

if __name__ == '__main__':
    juego = Game()
    gtk.main()
```



Con este código, cáda vez ques agregues un personaje a la lista `sprites`, se dibujará sobre la escena y recibirá actualizaciones.

Conforme el juego avance, es natural que encuentres formas las sofisticadas de administrar personajes, pero como un principio está bien!.

2.8 Finalizando... ¡No olvides ver la API!

Es importante comentar que tanto la clase `Sprite` cómo `Events` tienen mucha mas funcionalidad de la que comentamos aquí. Incluso hay muchísimas funcionalidades que no hemos siquiera comentado: colisiones, dibujado geométrico, animaciones, manejo de teclado etc...

Este tutorial es muy breve, su objetivo es ayudar a dar los primeros pasos con glucosa y explorar a grandes rasgos la API, pero hay mucho mas por conocer.

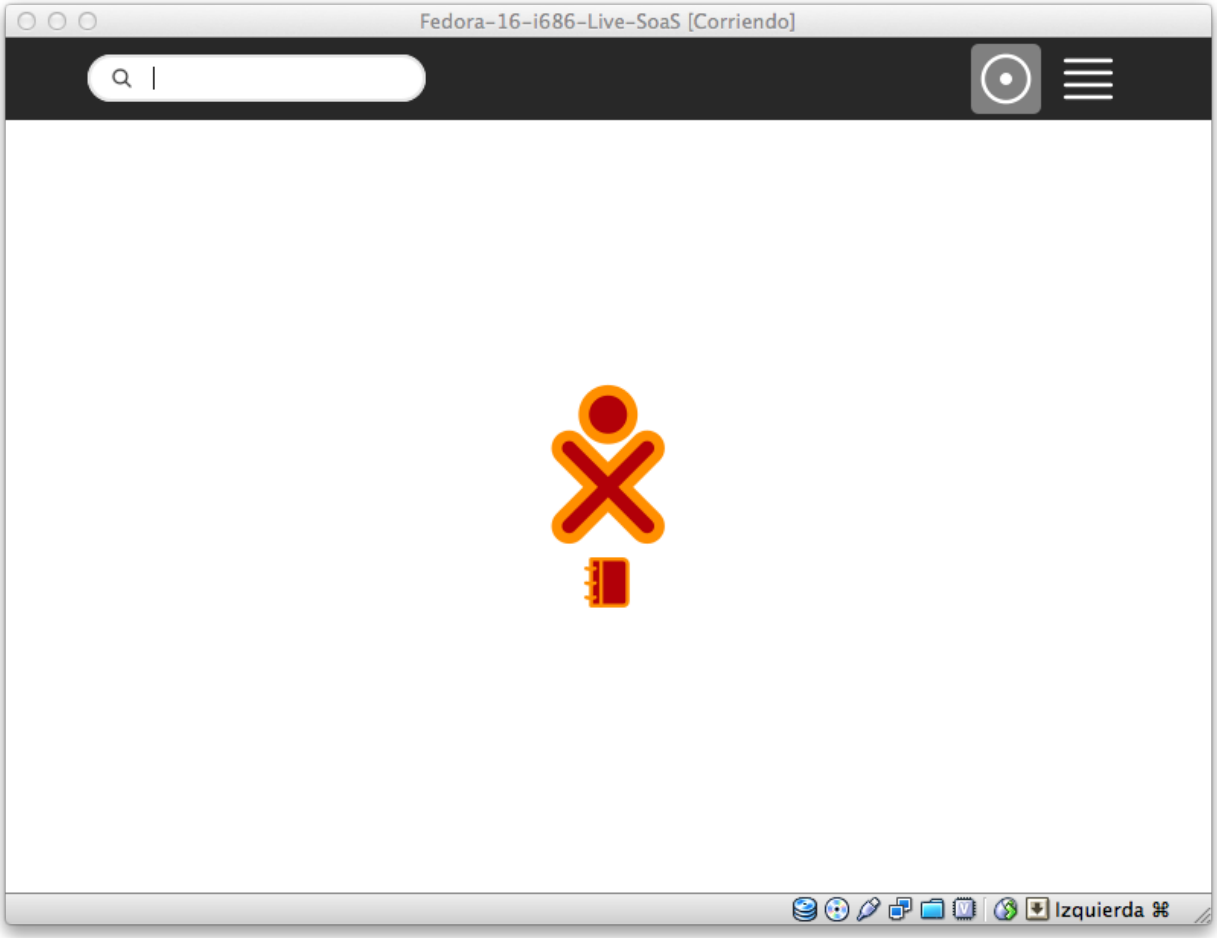
Para una descripción mas completa de toda la funcionalidad de glucosa, te recomendamos investigar la *API*, donde definimos todos los elementos que componen a glucosa y cómo utilizarlos.

CHAPTER 3

Cómo usar glucosa desde Sugar XO

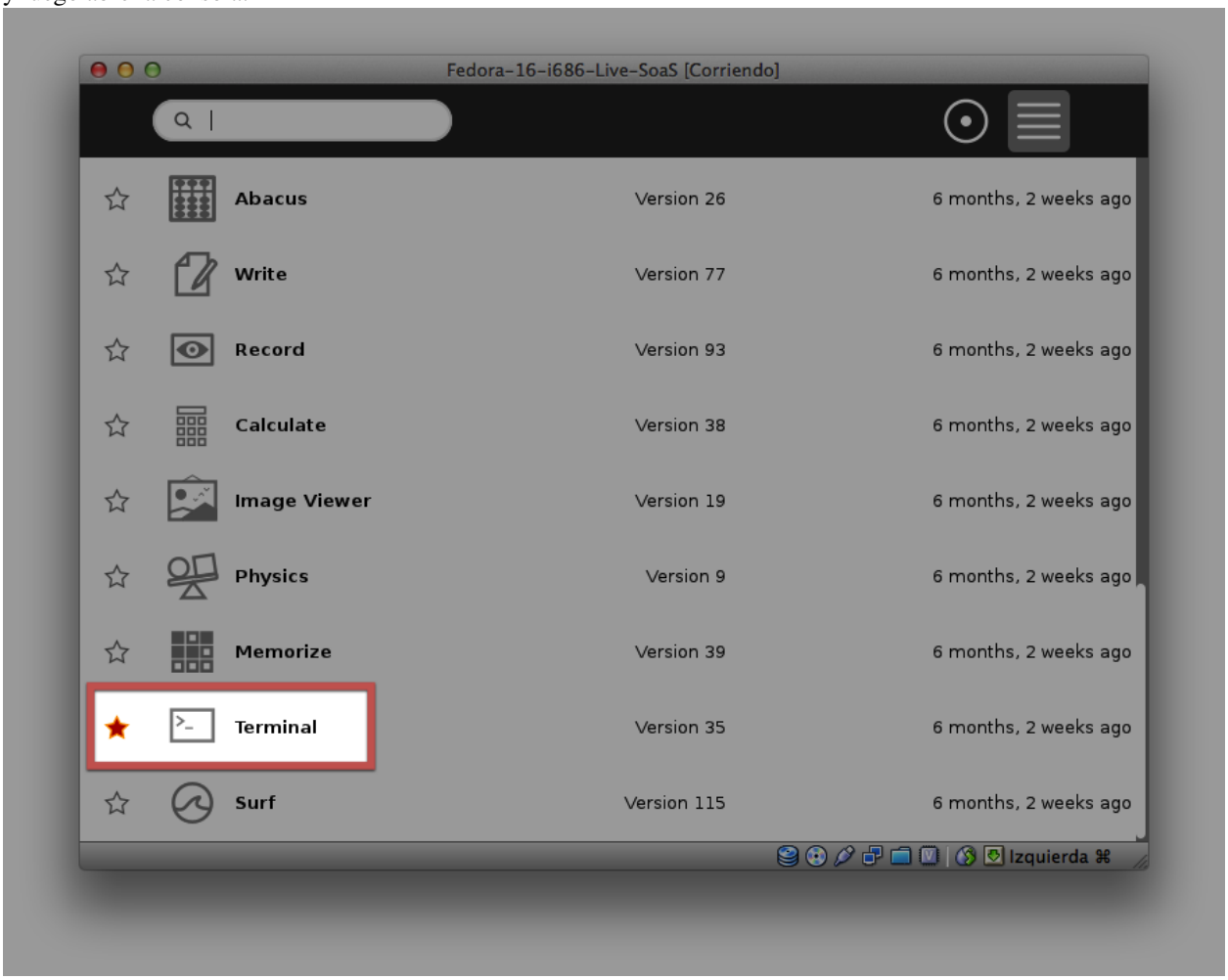
Para realizar puebas, y conocer la bilioteca, puedes descargar todo el código del repositorio y ejecutar nuestros demos.

Abre el entorno Sugar, ya sea desde una máquina XO, un emulador o máquina virtual. El proceso es el mismo en cualquier caso:



Una vez iniciado, elige la vista de lista:

y luego abre la consola:



Ahí, tienes que escribir algunos comandos para obtener todo el código de glucosa.

Si tienes `git` instalado, el comando para descargar glucosa es el siguiente:

```
git clone git://github.com/hugoruscitti/glucosa.git
```

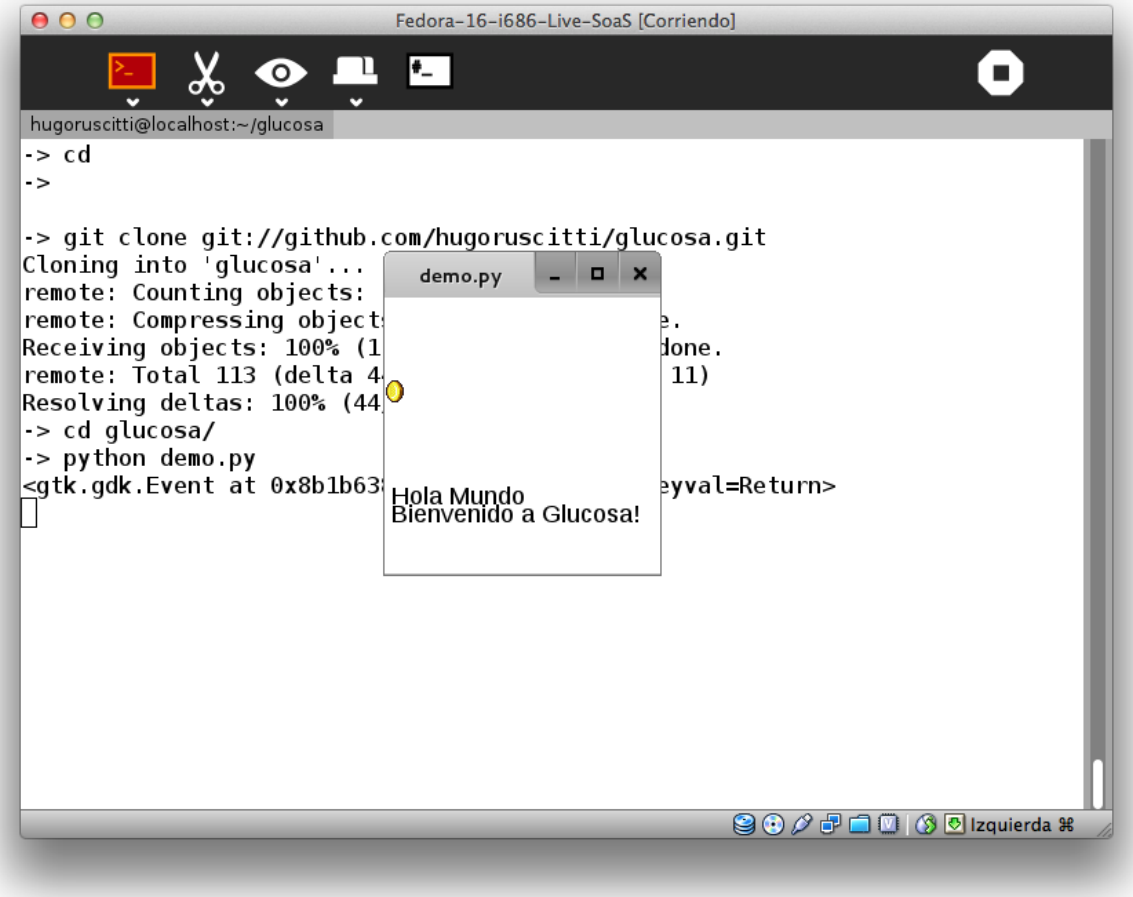
En caso contrario, utiliza el siguiente comando:

```
wget https://github.com/hugoruscitti/glucosa/tarball/master
tar xvzf master
```

Y luego, de cualquiera de las dos formas, solo tienes que ingresar en el directorio creado y ejecutar el script demostración:

```
cd glucosa
python demo.py
```

En la pantalla aparecerá un pequeño ejemplo indicado que todo a funcionado bien:



Fecilitaciones!

CHAPTER 4

Creando una actividad para Sugar XO

Glucosa incluye una plantilla de actividad para Sugar XO muy fácil de utilizar.

En esta sección veremos cómo crear una actividad básica a partir de la plantilla y cómo llevarla a la XO utilizando un pendrive.

4.1 Introducción

Las actividades de SugarXO tienen una estructura que le permiten a la XO realizar el proceso de despliegue, actualización y configuración correctamente.

El primer paso para crear una actividad es clonar el repositorio de glucosa:

```
git clone git://github.com/hugoruscitti/glucosa.git
```

Se creará un nuevo directorio llamado glucosa, y dentro de él un directorio con la plantilla de la actividad.

El siguiente paso es configurar la plantilla para que utilice glucosa y genere un repositorio propio:

```
cd glucosa
cd activity-template
cp ../glucosa.py ./

git init
git add .
git commit -m "creando actividad."
```

Los últimos tres comandos aseguran que el proceso de empaquetado de la aplicación recolecte todos los archivos necesarios. Si agregas nuevos archivos es importante hacer `git add .` y un `git commit -m "algo"` antes que empaquetar.

Note: En el directorio de la plantilla hay un archivo llamado `aceituna.png`, este archivo solo se utiliza en nuestro ejemplo, si quieres hacer una nueva actividad que no lo utiliza podrías borrarlo tranquilamente.

4.2 Generando la actividad

Una vez dentro del directorio de la plantilla puedes generar un paquete instalable de la actividad.

Tienes que ejecutar el siguiente comando:

```
python setup.py dist_xo
```

Y la nueva actividad se tendría que haber generado dentro del directorio `dist`:

```
$ ls dist/
ActivityTemplate-1.xo
```

Perfecto, ahora la aplicación se puede copiar en un pendrive y llevar a la XO:

```
cp dist/ActivityTemplate-1.xo /media/pendrive_path
```

4.3 Instalación

Solamente tienes que conectar el pendrive con la actividad directamente en el equipo XO:

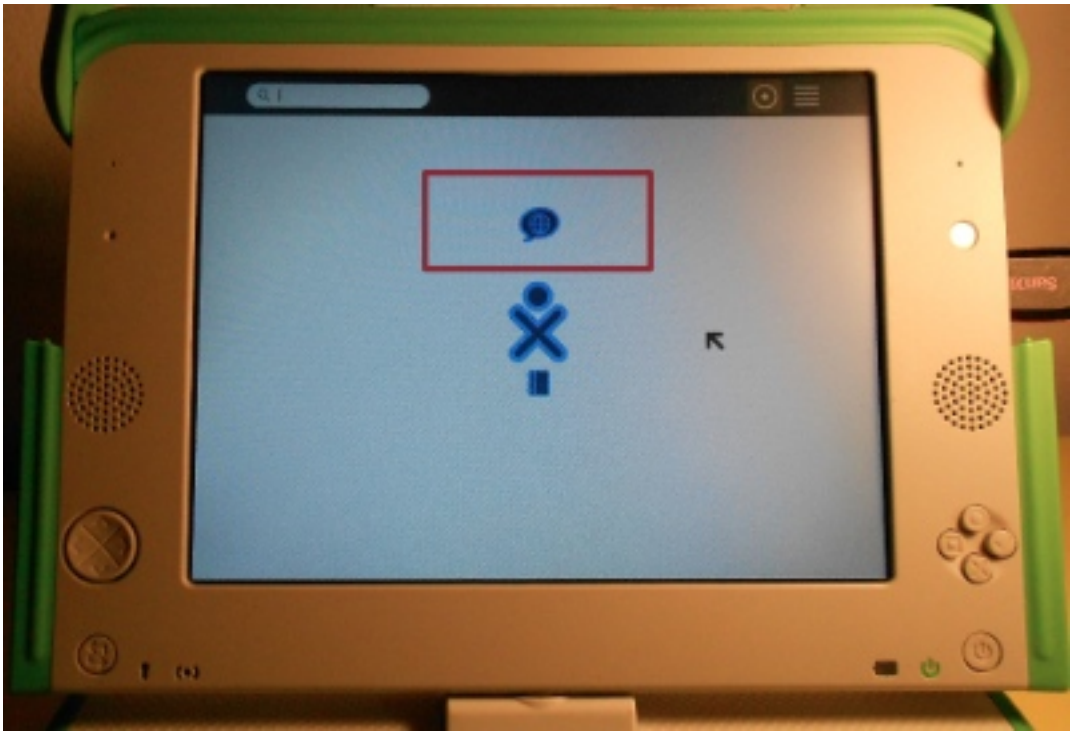
Cuando el pendrive ya esté conectado, tendría que aparecer en la parte inferior de la pantalla un ícono para acceder al contenido del pendrive:

Cuando lo selecciones, aparecerá el ícono de nuestra actividad, tienes que pulsarlo y eso iniciará la Instalación y ejecución de la actividad:

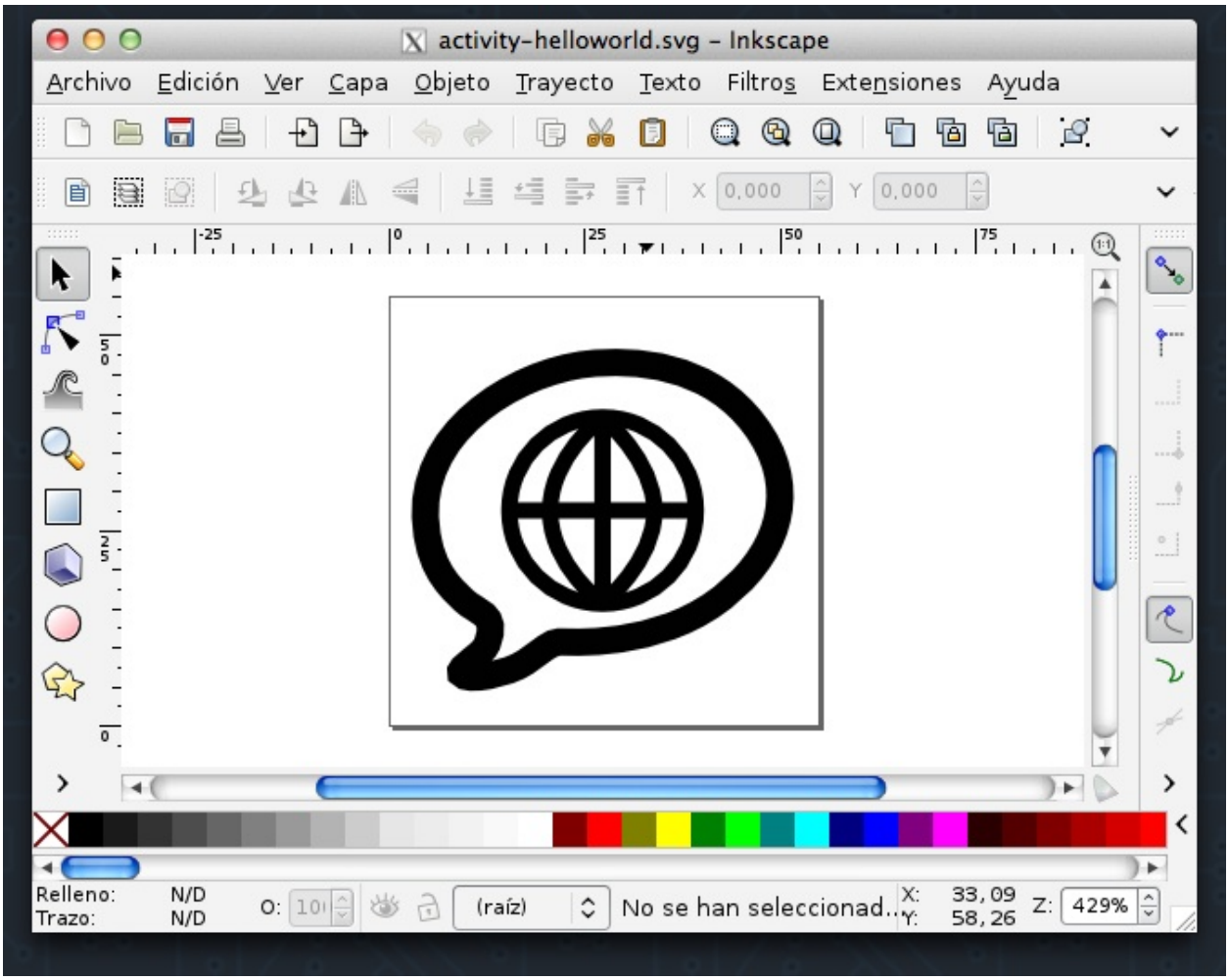
Funcionó !

4.4 Cambiando el ícono de la actividad

Cuando la actividad esté instalada, habrá un ícono que va a servir para identificarla en todo el sistema:



Este ícono se puede modificar, por ejemplo usando la aplicación Inkscape:



El archivo que represenata al icono es `activity/activity-helloworld.svg`.

4.5 Actualizando la actividad

Para actualizar una versión tienes que cambiar el código de tu actividad y llamar nuevamente al comando `python setup.py dist_xo`.

Es importante que actualices el archivo `activity/activity.info`, porque cada nueva versión tiene que aumentar el valor de la variable `activity_version`. De otra forma la XO no reconocerá que se trata de una nueva versión.

Otro punto a tener en cuenta, es que el proceso de empaquetado solamente tendrá en cuenta los archivos gestionados por el repositorio git que creamos al principio. Así que es una buena idea aprender algo de Git!, o simplemente hacer `git add .` y `git commit -m "?"` antes de realizar las nuevas vesiones.

4.6 ¿Dónde se instalan las actividades?

Las actividades se instalan en el directorio `Activities` (que está a su vez dentro del directorio personal del usuario).

Así que podrías acceder a ese directorio para hacer cambios de prueba o simplemente para borrar la actividad.

4.7 Distribuyendo la actividad

El equipo de desarrollo de sugar tiene un sitio espectacular para publicar y distribuir actividades. La dirección para acceder es:

http://activities.sugarlabs.org

CHAPTER 5

API

CHAPTER 6

Índices y tablas

- *genindex*
- *modindex*
- *search*