

Comunidad Pilas Engine

Aportes y resultados obtenidos

Zahradnicek Desirée

Ingeniería de Software de fuentes Abiertas/Libres

Participación en Comunidad

Cátedra:

- Ingeniería de Software de fuentes Abiertas/Libres

Docente:

- Medel, Ricardo Hugo

Alumna:

- Zahradnicek, Desirée

Legajo: 60003



UNIVERSIDAD TECNOLÓGICA NACIONAL

FACULTAD REGIONAL CÓRDOBA

- 2017 -

Índice

1. Objetivos, Servicios y componentes.....	5
¿Que es Pilas Engine?.....	5
Motores multimedia	5
Actores	6
Documentación	6
Repositorio	6
Obtener la última versión del repositorio.....	6
2. Comunidad	7
¿Cómo empezar?.....	7
Convocatoria a colaboradores	7
Licencia.....	8
Características técnicas	8
Cómo participar del desarrollo de pilas-engine	8
3. Experiencia Personal.....	9
Conclusión.....	14

Introducción

En el presente trabajo práctico para la Cátedra de Ingeniería de Software de fuentes Abiertas/Libres se busca poder llevar a la práctica el concepto de trabajo en comunidad, por tanto, se buscará ser participe en un proyecto real, siguiendo el mismo desde los primeros pasos en la comunidad hasta los avances obtenidos.

La comunidad elegida es Pylas Engine y a lo largo del presente conoceremos sus características, objetivos y trayectoria como así también el aporte realizado.

1. Objetivos, Servicios y componentes

¿Que es Pilas Engine?

Es un proyecto de software libre cuyo objetivo es permitir el fácil desarrollo de videojuegos a personas que no están familiarizadas con el desarrollo de los mismos. Por tanto, las bases son interfaces de programación que sean simples y fáciles de usar. Actualmente está diseñada por desarrolladores de habla hispana y cuenta con el respaldo de docentes y profesionales especialistas en desarrollo de Software. El lenguaje de programación utilizado es Python, y se busca aprovechar su modo interactivo.

Las bibliotecas que utilizan son dos: Box2D, como motor de física, mientras que Qt es un motor multimedia utilizado para dibujar, reproducir sonidos y manejar eventos. Por otro lado, utilizan solo 3 componentes indispensables: Mundo, Actor, Motor

Mundo es un objeto `singleton`, hay una sola instancia de esta clase en todo el sistema y se encarga de mantener el juego en funcionamiento e interactuando con el usuario.

Los actores (clase `Actor`) representan a los personajes de los juegos, la clase se encarga de representar todos sus atributos como la posición y comportamiento como "dibujarse en la ventana" (Sprites).

El Motor, permite soporte multimedia portable y multiplataforma. Se delega la tarea de dibujar, emitir sonidos y controlar eventos a una biblioteca externa, la misma es Qt, pero en versiones anteriores ha sido implementada en pygame y sfml.

Motores multimedia

Al principio pilas delegaba todo el manejo multimedia a una biblioteca llamada SFML. Pero esta biblioteca requería que todos los equipos en donde funcionan tengan aceleradoras gráficas (al menos con soporte OpenGL básico).

Se reemplazó el soporte multimedia con la biblioteca Qt. Que sabe acceder a las funciones de aceleración de gráficos (si están disponibles), o brinda una capa de compatibilidad con equipos antiguos.

La función que permite iniciar y seleccionar el motor es `pilas.iniciar`.

```
pilas.iniciar(usr_motor='qt')
```

Esto funciona gracias al polimorfismo; el objeto motor sabe que tiene que delegar el manejo multimedia a una instancia (o derivada) de la clase Motor (ver directorio pilas/motores/).

El motor expone toda la funcionalidad que se necesita para hacer un juego: sabe crear una ventana, pintar una imagen o reproducir sonidos, entre tantas otras cosas.

El objeto mundo no sabe exactamente que motor está utilizando, solo tiene una referencia a un motor y delega en él todas las tareas multimedia. Solo puede haber una instancia de motor en funcionamiento, y se define cuando se inicia el motor.

Actores

Los actores permiten que los juegos cobren atractivo, porque un actor puede representarse con una imagen en pantalla. La implementación de todos los actores están en el directorio pilas/actores. Todos los actores heredan de la clase Actor, que define el comportamiento común de todos los actores.

Hay dos métodos en los actores que se invocarán en todo momento: el método `actualizar` se invocará cuando el bucle de juego del mundo llame al método `_realizar_actualizacion_logica`, esto ocurre unas 60 veces por segundo. Y el otro método es `dibujar`, que también se invoca desde el objeto mundo, pero esta vez en el método `_realizar_actualizacion_grafica`.

Documentación

El sistema de documentación usado es Sphinx, el mismo permite gestionar todo el contenido del manual en texto plano, y gracias a varias herramientas de conversión como restructuredText y latex, se producen muchos formatos de salida como HTML y PDF. Toda la documentación del proyecto está en el directorio doc. El directorio doc/sources contiene todos los archivos que se modifican para escribir contenido en la documentación. Para generar los archivos PDF o HTML se usa el comando `make` dentro del directorio doc. El archivo que dispara todas las acciones que sphinx sabe hacer están definidas en el archivo Makefile.

Repositorio

Para contribuir en el desarrollo se necesita una cuenta de usuario en [github](https://github.com). La dirección de acceso web al repositorio es: <http://github.com/hugoruscitti/pilas>

Obtener la última versión del repositorio

Se tiene que ejecutar el siguiente comando desde un terminal: `git clone http://github.com/hugoruscitti/pilas`. Luego aparece un directorio llamado pilas, con el contenido completo del repositorio.

2. Comunidad

El equipo de Pilas Engine está conformado por varios miembros, basado en principios de una comunidad de software libre, entre los mismos se pueden encontrar desarrolladores, docentes y estudiantes. Más allá de ser una comunidad de habla hispana, sus colaboradores se distribuyen por todo el mundo.

Equipo principal



Hugo Ruscitti



Walter Velazquez



Quique Porta



Fernando Salamero



Irving Rodríguez

Colaboradores:



Marcos Vanetta



Luciano Baraglia



Hernantz



Pablo Mouzo



Diego Accorinti



JuanBC



binary-sequence



José Luis Di Biase



Felipe González



Ivan Pedrazas



Jairo Trad



Matías Iturburu



Diego Riquelme

¿Cómo empezar?

Se puede instalar todo el kit de desarrollo siguiendo las instrucciones de la web: <http://www.pilas-engine.com.ar>. _Una vez instalada la biblioteca, se puede invocar al comando pilas -e para ver una lista completa de ejemplos y mini juegos.

Convocatoria a colaboradores

Para poder conocer las maneras de participar en la comunidad, la misma cuenta con una página: <https://github.com/hugoruscitti/pilas/wiki/participar> donde se ayuda con breves descripciones a adentrarse en el mundo de Pilas Engine. Se puede colaborar del proyecto y participar en el desarrollo.

Licencia

Pilas se distribuye bajo la licencia LGPLv3.

Características técnicas

- Es multiplataforma (Funciona sobre GNU/Linux y Windows)
- Cuenta con objetos prediseñados para agilizar el desarrollo.
- Tiene documentación completamente en español.
- Se utiliza desde Python, lo que permite usarla desde sesiones interactivas.
- Incluye interacción con el motor de física pybox2d.
- Es software libre.
- Cuenta con varios ejemplos y tutoriales para ir paso a paso.

Cómo participar del desarrollo de pilas-engine

Básicamente participar en Pilas Engine no requiere de mucho esfuerzo ni de aceptación ni de protocolos que cumplir, como todo proyecto de software libre, los desarrollos son el resultado de trabajo en forma conjunta y colaborativa en comunidad. Para comenzar (en mi caso personal), se debe buscar una tarea interesante en la que se tenga interés en participar.

Alternativas

Participación en foros:

Se debe dar de alta una cuenta en el foro (<http://foro.pilas-engine.com.ar>), presentarse y ayudar a resolver preguntas de otros usuarios. También se puede aportar ideas, conversar, y publicar juegos.

Mejorar el manual

El manual de pilas-engine está disponible en github (<https://github.com/hugoruscitti/pilas-manual>), se puede clonar y publicar cambios libremente. Este repositorio genera de forma automática el manual de pilas y la versión web que está en: <http://hugoruscitti.github.io/pilas-manual/>

Corrección de bugs y mejoras

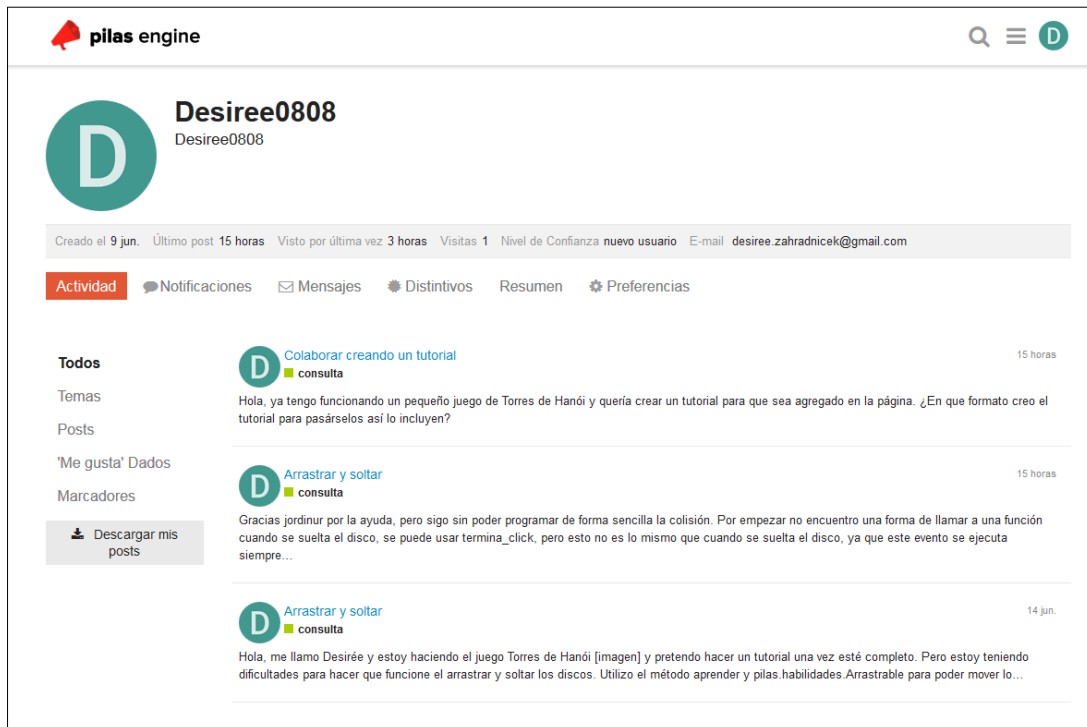
Nos podemos encontrar con un listado de tareas que se pueden llevar a cabo junto con mejoras, se ubican en la sección de Issues, las mismas no han sido asignadas aun y por lo tanto si uno se interesa por llevarlas a cabo se puede enviar un pull request a: <https://github.com/hugoruscitti/pilas>

Nuevos cursos o tutoriales

Se necesita de aportantes con ganas de llevar a cabo tutoriales, juegos y cursos sobre pilas-engine. Una vez que el tutorial se publica, Pilas pide permiso al autor para subirlo a la web (en la sección documentación).

3. Experiencia Personal

Para empezar, tuve que registrarme en el foro que posee la comunidad; luego de conocer los tópicos y necesidades que existían realicé un posteo en el mismo donde explicaba las intenciones de aporte que quería realizar.



Casi no hubo demoras de tiempo entre que se realizaron las preguntas en el post y se obtuvieron las respuestas, teniendo en cuenta que los posteos son moderados y deben pasar por una evaluación (desconozco quien se encarga de esto)

El usuario Jordinur respondió dudas con amabilidad y mucha paciencia y compromiso y se generó un Feedback muy interesante de las dudas que fueron surgiendo.

Por un lado, en Github se encuentran los que desarrollan el software y por otro lado el foro (tanto para desarrollo y hacer juegos, (quizá más pensado para los que quieren hacer juegos, pero de ahí sacan peticiones de modificaciones y cosas por el estilo)).

En Github no participé nada si no en el foro para hacer un tutorial, se buscó un juego distinto de ingenio de mesa que busca en definitiva usar el software como herramienta para hacer algo y no con lo que ya trae, se buscó hacer algo propio. Es un tutorial no exclusivo de Pilas, si no que usa Pilas para graficar y todos los entornos sencillos, pero la lógica está programada desde cero.

La propuesta que se llevó a cabo en la medida de lo posible es el desarrollo del Juego Torres de Hanói, el cual es un rompecabezas o juego matemático que consiste en un número de discos de radio creciente que se apilan insertándose en una de las tres estacas de un tablero. El objetivo del juego es crear la pila en otra de las estacas siguiendo ciertas reglas. Por otro lado, también escribir el tutorial del juego creado para poder aportarlo a la comunidad.

Actualmente el juego se encuentra terminado y aportado en el foro a la comunidad y el manual no está completo en su totalidad por lo tanto el mismo no se ha cargado.

Enlace: <http://foro.pilas-engine.com.ar/t/arrastrar-y-soltar/1107/5>

Arrastrar y soltar

 consulta



Desirée0808

Hola, me llamo Desirée y estoy haciendo el juego Torres de Hanói

1  7d



y pretendo hacer un tutorial una vez esté completo. Pero estoy teniendo dificultades para hacer que funcione el arrastrar y soltar los discos. Utilizo el método aprender y pilas.habilidades.Arrastrable para poder mover los discos, pero necesito de alguna forma que al soltar el disco se ejecute una función, la cual debe encargarse de detectar si hay contacto (colisión) con uno de los postes y de ser así registrar el movimiento. En concreto tengo dos problemas:


1. Detectar cuando se suelta un disco
2. Detectar la colisión entre la pieza soltada y los 3 posibles postes

Paso el código que tengo funcionando hasta ahora (con solución automática incluida 😊), para realizar movimientos manuales por ahora hay que llamar a la función realizar_movimiento(origen,destino), la cuál solo realiza el movimiento si éste es válido.

```
# coding: utf-8
import pilasengine

pilas = pilasengine.iniciar()

# Número de discos del juego
numero_discos = 5
```






Arrastrar y soltar

consulta

Desde ya muchas gracias y felicitaciones por tan interesante proyecto!

1 Respuesta




Responder

creado

D 7d


última respuesta


D 15h


4
respuestas

44
visitas

2
usuarios






jordinur
5d

Buenas, @Desiree0808 !

Sin duda un trabajo impresionante... 🤖

Aunque debo decir que tus conocimientos a buen seguro superan los míos, me voy a tomar la libertad de sugerirte que programes unas colisiones entre el disco que está siendo arrastrado y los postes.

Al realizarse la colisión debería ejecutarse la función

D Desiree0808:

realizar_movimiento(origen,destino)

Esta función ya la tienes programada correctamente, por lo que no debería haber mayor problema.




Puedes consultar el manual sobre el tema de las colisiones, donde seguro encontrarás información.

Quizás sería más fácil detectar esas colisiones si palos y discos fueran actores diferentes (al menos es como yo lo hubiera probado en un principio)

Si aún así sigues teniendo dudas dínoslo y yo mismo intentaré programar unas colisiones entre los discos y las barras en un ejemplo sencillo partiendo de tu código.

Muchas gracias por tu interés y tu tiempo, y este maravilloso juego que estás creando! 😊

Un abrazo.




Responder

Consulta completa en el foro y ayuda:

<http://foro.pilas-engine.com.ar/t/arrastrar-y-soltar/1107>

Imagen del juego resultante:



A modo de resumen se explican los métodos y funciones principales:

Se deben setear el

- Número de discos del juego
- Estado del juego: El estado del juego se guarda en una lista de tres componentes, donde cada una corresponde a cada uno de los postes y cuyo valor es a su vez también una lista, que contiene los números de los discos que se encuentran en ese poste. Los discos se numeran de 0 a numero_discos desde el más grande hasta el más pequeño.
- Estado inicial del juego
- Número de movimientos realizados

Creaciones:

- Crear los postes y base
- Crear los discos

Lógica del juego

- Validar movimiento: Esta función toma como entrada dos valores, el número de poste desde el cual se saca el disco y el número de poste en el cual se quiere poner dicho disco, y devuelve True si es un movimiento válido y False si no lo es.
- Realizar movimiento: Esta función toma como entrada dos valores, el número de poste desde el cual se saca el disco y el número de poste en el cual se quiere poner dicho disco, y en caso de que el movimiento sea válido lo realiza
- Verificar fin de juego: Esta función verifica el estado del juego y devuelve True si el juego ha sido resuelto y False en caso contrario.
`def verificar_fin_de_juego():` for poste in juego[1:]: # todos los postes menos en el inicial (0) if len(poste) == numero_discos: todos

los discos están en este poste, si todos los movimientos fueron válidos entonces la torre debe estar completa y el juego ha sido resuelto, no obstante vamos a verificar que los discos estén en el orden correcto.

Conclusión

El presente trabajo práctico ha sido de mucha ayuda para poder entender las interacciones de una comunidad de software libre y los pilares que la sostienen; la colaboración y ayuda han primado en todo momento, frente a diversas dudas que fueron surgiendo, las mismas han sido resueltas por diferentes miembros y el uso de diversas herramientas y foros ha hecho la diferencia.

Pilas Engine para mí resultó ser una experiencia muy grata, ya que no tuve frenos para poder llevar a cabo un aporte de mi parte, por tanto, desde el primer momento obtuve ayuda y se pudieron ver resultados en ello. Sin duda me encuentro en momento de poder recomendar esta actividad práctica para con una comunidad de software libre y también una cátedra como esta donde nos ofrecieron una perspectiva diferente a la que tenemos desde el ingreso a la facultad.