

Universidad de Deusto  
Ingeniería en Informática  
Proyecto Fin de Carrera

**Tennacles**

Grzegorz Adam Hankiewicz (78930020)

Director: Iker Jamardo Zugaza

Mayo de 2003



 <p>Universidad de Deusto • • • • Facultad de Ingeniería</p>		Proyecto Fin de Carrera
---	--	----------------------------


## Resumen

Tennacles es una aplicación de diseño de aventuras gráficas. El objetivo es facilitar la tarea de reunir los recursos necesarios como gráficos, sonido, lógica y componerlos y planificar la aventura de una forma genérica sin importar el motor gráfico que vaya a ejecutarla. Tennacles se perfila como una herramienta de desarrollo asistido, similar en algunos momentos a otras herramientas conocidas de autor para la creación de aplicaciones multimedia como Director de Macromedia.

## Descriptores

Multimedia, compiladores, gramáticas, teoría de lenguajes




 <p>Universidad de Deusto Facultad de Ingeniería</p>	<p>Índice general</p>	<p>Proyecto Fin de Carrera</p>
---	-----------------------	--------------------------------

# Índice general

<b>1. Documento de objetivos</b>	<b>1</b>
1.1. Introducción . . . . .	1
1.2. Materializando ideas y conceptos . . . . .	2
1.3. Licencia . . . . .	2
1.4. Diseño general de Tennacles . . . . .	3
1.5. Módulo de personajes y objetos . . . . .	5
1.6. Módulo de escenas . . . . .	5
1.6.1. Capa de fondo . . . . .	5
1.6.2. Capa de Z-buffer . . . . .	6
1.6.3. Capa de desplazamiento . . . . .	6
1.6.4. Capa de zonas calientes . . . . .	6
1.6.5. Capa de actores . . . . .	6
1.7. Módulo de conversaciones . . . . .	6
1.8. Módulo de problemas lógicos . . . . .	7
1.9. Plugin de exportación Tennacles -> Mad . . . . .	7
<b>2. Justificación del proyecto</b>	<b>9</b>
2.1. Videojuegos . . . . .	9
2.1.1. Evolución . . . . .	9
2.1.2. Hechos actuales . . . . .	10
2.1.3. El género de las aventuras . . . . .	11
2.1.4. Objetivo de Tennacles . . . . .	12
2.2. Python . . . . .	13
2.2.1. ¿Qué es? . . . . .	13
2.2.2. Toma inicial de contacto . . . . .	14
2.2.3. Tipos de datos . . . . .	15
Listas . . . . .	15
Tuplas . . . . .	16
Diccionarios . . . . .	16
2.2.4. Características avanzadas . . . . .	17
Funciones anónimas . . . . .	17
Todo es una interfaz . . . . .	17



Iteradores y generadores . . . . .	18
Acceso controlado a los atributos . . . . .	19
Gran librería estándar . . . . .	20
2.2.5. Otras opiniones sobre Python . . . . .	21
2.3. GTK y PyGTK . . . . .	22
2.4. Allegro y otras librerías complementarias . . . . .	24
<b>3. Diseño de las ventanas de la interfaz</b>	<b>27</b>
3.1. Bienvenida . . . . .	27
3.2. Ventana de gestión de recursos . . . . .	28
3.2.1. Menú “Repositorio” . . . . .	28
3.2.2. Menú “Nuevo recurso” . . . . .	29
3.2.3. Menú “Ayuda” . . . . .	29
3.3. Atributos del proyecto . . . . .	29
3.4. Selección de plugin . . . . .	30
3.5. Editor de escenas . . . . .	30
3.6. Ventana de atributos . . . . .	32
3.7. Editor de acciones lógicas . . . . .	32
3.7.1. Condiciones . . . . .	32
3.7.2. Acciones . . . . .	33
3.8. Editor de actores . . . . .	33
3.9. Editor de diálogos . . . . .	33
<b>4. Tipos de recursos</b>	<b>35</b>
4.1. Recursos físicos . . . . .	36
4.1.1. Ficheros de imágenes . . . . .	36
4.1.2. Ficheros de audio . . . . .	37
4.1.3. Ficheros de texto . . . . .	37
4.2. Recursos lógicos . . . . .	38
4.2.1. Sprites . . . . .	39
4.2.2. Sonidos . . . . .	41
4.2.3. Acciones lógicas . . . . .	41
4.2.4. Líneas temporales . . . . .	42
4.2.5. Animaciones . . . . .	42
4.2.6. Actores . . . . .	43
4.2.7. Escenas . . . . .	44
<b>5. Otras clases</b>	<b>45</b>
5.1. Proyecto . . . . .	45
5.2. Autor . . . . .	45
5.3. Escena . . . . .	46
5.4. Ventanas de interfaz . . . . .	46
<b>6. Diccionario de datos</b>	<b>49</b>
6.1. Nodo raíz . . . . .	49
6.2. Nodo autor . . . . .	49
6.3. Nodo recursos . . . . .	49
6.4. Nodo imagen . . . . .	50

 Universidad de Deusto Facultad de Ingeniería	Índice general	Proyecto Fin de Carrera
---	----------------	-------------------------

6.5. Nodo audio . . . . .	50
6.6. Nodo texto . . . . .	50
6.7. Nodo sprite . . . . .	50
6.8. Nodo sonido . . . . .	51
6.9. Nodo accion_logica . . . . .	51
6.10. Nodo linea_temporal . . . . .	51
6.11. Nodo frame . . . . .	51
6.12. Nodo actor . . . . .	51
6.13. Nodo animacion_ir . . . . .	52
6.14. Nodo animacion_usar . . . . .	52
6.15. Nodo animacion_hablar . . . . .	52
6.16. Nodo animacion . . . . .	52
6.17. Nodo escena . . . . .	52
6.18. Nodo capa . . . . .	53
6.19. Ejemplo de un fichero XML . . . . .	54
<b>7. Algoritmos complejos</b>	<b>57</b>
7.1. Actualizar recursos . . . . .	57
7.2. Salvar repositorio . . . . .	58
7.3. Dibujar recursos . . . . .	58
7.4. Controlando el editor de escenas . . . . .	58
<b>8. Interfaz de Allegro con PyGTK</b>	<b>61</b>
<b>9. Interfaz de Tennacles con plugins exportadores</b>	<b>65</b>
9.1. API exportada por los plugins . . . . .	65
9.2. API exportada por Tennacles . . . . .	66
<b>10. Mejoras posibles</b>	<b>69</b>
<b>11. Conclusión</b>	<b>73</b>
<b>Acrónimos</b>	<b>75</b>
<b>Bibliografía (Libros y artículos)</b>	<b>79</b>
<b>Bibliografía (Otras fuentes documentales)</b>	<b>81</b>







# Índice de figuras

2.1. Dependencias de software. . . . .	25
4.1. Jerarquía de clases: recursos físicos. . . . .	36
4.2. Jerarquía de clases: recursos lógicos simples. . . . .	39
4.3. Jerarquía de clases: recursos lógicos complejos. . . . .	40
5.1. Otras clases. . . . .	46
8.1. Esquema de traducción del “Enlace” o “binding” entre los lenguajes Python y C. . . . .	62



 <p>Universidad de Deusto • • • • Facultad de Ingeniería</p>	1 Documento de objetivos	Proyecto Fin de Carrera
---	--------------------------	----------------------------

# CAPÍTULO 1

## Documento de objetivos

### 1.1 Introducción


---

Hoy en día para crear una aventura gráfica es necesario un equipo formado por programadores y artistas: los primeros escriben código, que será llamado “motor” del juego, y los últimos crean los gráficos que verá el usuario. Cada aventura requiere nuevos gráficos que serán creados desde cero por el equipo gráfico. No obstante, siguiendo una orientación de diseño genérica, es posible para los programadores crear un motor cuya principal característica sea la reusabilidad, lo que significaría que para producir la siguiente aventura, el primer equipo no necesitaría comenzar desde cero (lo cual no debería ocurrir si el motor es escrito correctamente la primera vez).

Esto ya ha ocurrido, y empresas como LucasArts [24] han creado motores como SCUMM que han sido usados sin grandes cambios (normalmente sólo aquellos necesarios para adaptar el motor al nuevo hardware o para añadir características imposibles en el pasado) a lo largo del ciclo de vida de la compañía, reusándolo múltiples veces. Hay incluso emuladores o máquinas virtuales que permiten reproducir los juegos publicados en plataformas para las cuales no fueron diseñados.

A pesar de haber muchos proyectos cuyo objetivo principal es crear un motor sólido que puede ser usado para reproducir estos juegos, en la práctica no existen proyectos centrados en el desarrollo de una herramienta que pueda ser usada por el equipo de artistas para crear los datos necesarios por un motor. Algunos motores son acompañados de herramientas básicas que ayudan en la construcción de aventuras, pero no están orientadas hacia artistas con poca o ninguna experiencia en programación.

El resultado final es que la mayoría de los juegos de aventuras gráficas que no son creados por grandes compañías, están escritos por grupos de programadores que son capaces de usar el motor sin problemas, pero que suelen tener poca habilidad artística, necesaria para crear un producto final atractivo para el usuario medio. Incluso en los casos en los que el motor tiene una herramienta parcialmente para ayudar en el desarrollo de la aventura gráfica, está orientada a artistas gráficos que saben cómo funcionan los videojuegos

 <p>Universidad de Deusto Facultad de Ingeniería</p>	<p>1 Documento de objetivos</p>	<p>Proyecto Fin de Carrera</p>
---	---------------------------------	--------------------------------

internamente y/o tienen las habilidades necesarias para programar el comportamiento de los personajes del juego (normalmente llamada IA, inteligencia artificial).

Este proyecto universitario intentará resolver esta parte del problema del desarrollo de aventuras gráficas: los juegos deberían ser diseñados y dibujados, no programados [15]. Se pretende proveer las herramientas visuales capaces de ensamblar y producir los datos que posteriormente serán reproducidos por un motor existente de aventuras gráficas, proporcionando un entorno de desarrollo integrado controlado fácilmente por un artista gráfico o diseñador que no necesitará tocar código del motor. En otras palabras: el usuario de la herramienta será capaz de crear una aventura gráfica jugable desde cero, y reduciendo el tiempo de desarrollo necesario.

## 1.2 Materializando ideas y conceptos


La herramienta desarrollada durante este proyecto (de ahora en adelante llamada Tennacles), será implementada en Python [36], un lenguaje interpretado de alto nivel. Para la interfaz de Tennacles se usará el grupo de componentes gráficos GTK 2.0 [12]. Para la representación/reproducción de los gráficos y sonidos se usará la librería multimedia de programación de videojuegos Allegro [1]. Al usar este software se asegura que Tennacles será portable a múltiples plataformas, aquellas soportadas simultáneamente por las herramientas mencionadas, principalmente Windows y Linux.

Tennacles almacenará los datos de las aventuras gráficas usando un formato de fichero con estructura interna propia, diferente del formato usado por los motores de las aventuras gráficas. Esta independencia es intencionada, ya que permitirá que los “datos fuentes” de la aventura estén aislados de los posibles cambios que se realicen en los motores. Tennacles trabajará directamente con los datos (gráficos, sonido y texto) creados por los artistas y guionistas, lo que significa que el autor podrá crearlos con la herramienta que conozca o se ajuste a sus necesidades. El desarrollo de una aventura gráfica con Tennacles será equivalente a ensamblar los datos creados por los artistas, los cuales serán convertidos al formato específico de cada motor más tarde por un software externo llamado “plugin exportador”. Tennacles solamente permitirá el desarrollo de aventuras que funcionen con motores gráficos bidimensionales, quedando las aventuras que usen representación tridimensional excluidas.

Aparte de las herramientas mencionadas, Tennacles será desarrollado completamente usando un sistema operativo libre, GNU/Linux Debian [4]. El desarrollo del proyecto se llevará a cabo a través de la web Savannah [29]. Savannah es un punto central de desarrollo, distribución y manutención de software libre. Aparte de las posibilidades de alojamiento de páginas web (prácticamente ilimitado), Savannah también ofrece otros servicios, como CVS [3], listas de correo, monitores de fallos/parches, gestores de tareas y publicación de ficheros. La característica más importante es el CVS, un sistema de versionado concurrente on-line, que será usado desde el comienzo del proyecto para poder llevar un histórico de los cambios realizados sobre código fuente de Tennacles y cualquier otra documentación (como la que está leyendo).

## 1.3 Licencia

La licencia usada por el proyecto es GPL [11]. Las razones para usar esta licencia, comenzando por la más importante, son:


 <p>Universidad de Deusto Facultad de Ingeniería</p>	<h2>1.4 Diseño general de Tennacles</h2>	<p>Proyecto Fin de Carrera</p>
--	--	--------------------------------

- El software debe ser libre. Este es uno de los principios de la ética del hacker [13, 28, 14], la verdadera guía para personas como yo, que les gusta investigar, experimentar y aprender. Todo esto es aplicable al entorno universitario al cual deberían estar expuestos los estudiantes: ¿cómo puede un programador aprender nuevas técnicas de programación si éstas fueron patentadas absurdamente por un monopolio? ¿Por qué debe estar la información restringida a otros programadores? Sin acceso completamente libre a la información, los programadores del mundo están condenados a “reescribir la rueda”, debido a la avaricia de alguien. Además, el libre intercambio de información, particularmente cuando la información está en forma de programas de ordenador, permite desarrollar una mayor creatividad global.
- Relacionado con el punto anterior, mi propio trabajo debe ser libre. Ya estoy en deuda con todas las personas que han contribuido al software que usaré durante este proyecto. Por lo tanto, liberaré mi propio trabajo para que otros puedan mejorarlo o usarlo como base de sus propios proyectos. Esto es lo menos que puedo hacer para expresar mi gratitud.
- Las herramientas como Tennacles no tienen sentido como software cerrado. En ejemplos previos de liberación de código fuente bajo licencias GPL, puede observarse cómo John Carmack, creador de la serie de juegos “Doom” y “Quake”, ha liberado el código fuente de los motores usados en sus populares juegos. No obstante, los gráficos y los niveles siguen sin ser libres. Esto es porque los juegos no son los motores ni son las herramientas que fueron usadas para crearlos. Los juegos son gráficos, los juegos son guiones interesantes, los juegos son jugabilidad. Y todas estas características no tienen nada que ver con motores o herramientas. Al liberar este tipo de productos al público se incrementa la posibilidad de que otros puedan crear juegos con ellos, lo que es de hecho el objetivo de este proyecto, crear una herramienta útil para otras personas.
- Usar una licencia como la GPL me permite incorporar a mi propio proyecto código GPL ya escrito. Dada la existencia de código que ha sido modularizado, mejorado, verificado y extendido, usar código GPL en este proyecto aumentará la velocidad de desarrollo. En la vida real esto equivale a tiempos de desarrollo menores y a reducir el coste global de un proyecto.
- Usar la licencia GPL me permite beneficiarme económicamente de mi trabajo. A pesar de que la GPL permite la libre modificación y duplicación del código, me permite también vender el código a otros (mientras éstos tengan los mismos derechos que he tenido yo), lo cual me permite en teoría crear en un futuro próximo una empresa dedicada al desarrollo de Tennacles (y otras herramientas relacionadas), contribuyendo al mundo con código y obteniendo suficiente dinero como para vivir de su desarrollo.

## 1.4 Diseño general de Tennacles

---

Tennacles es una integración, un pequeño grupo de herramientas, cada una de ellas diseñada específicamente para completar una tarea necesaria para crear una aventura gráfica. Los módulos que compondrán Tennacles serán los siguientes:

 <p>Universidad de Deusto Facultad de Ingeniería</p>	<p>1 Documento de objetivos</p>	<p>Proyecto Fin de Carrera</p>
---	---------------------------------	--------------------------------

### **Creación y diseño de escenas**

Las escenas son los fondos sobre los cuales se moverán los personajes y objetos. Básicamente Tennacles permitirá al usuario ver siempre la escena del juego, y sus acciones serán modificar la escena o sus objetos. Por esta razón es uno de los módulos más importantes de la herramienta.

### **Creación y diseño de personajes y objetos**

Los personajes y objetos deben ser creados a partir de sprites (imágenes) y otros datos adicionales como posición, momento en el que aparecen en pantalla, etc. Para simplificar esta tarea, el módulo permitirá la creación de un nuevo tipo de recurso (personaje u objeto) a partir de otros recursos más simples (principalmente gráficos). Esto significa seleccionar diferentes gráficos para componer una animación, ya que cada personaje u objeto puede tener diferentes animaciones para hablar, manipular la escena, etc.

### **Creación y diseño de conversaciones**

Uno de los puntos más interesantes de las aventuras gráficas es la interacción con otros personajes no controlados por el jugador (de ahora en adelante PNJ, personajes no jugadores), para darles objetos, preguntarles por información, etc. Con este módulo el usuario será capaz de crear árboles de conversación que permitirán tener una conversación coherente predeterminada. Las ramas de los árboles pueden tener bucles que a su vez pueden estar condicionados por otras variables lógicas del juego.


### **Creación y diseño de problemas lógicos**

Desde un punto de vista abstracto, la mayoría de los problemas que se pueden encontrar en una aventura gráfica y que deben ser solucionados por el usuario se pueden explicar como una secuencia de pasos/acciones necesarias para llegar al resultado final. Estos pasos/acciones y sus resultados se pueden modelar en este módulo como variables. El usuario será capaz de interconectar variables y crear condiciones complejas que le permitirán crear bifurcaciones en el argumento de la aventura gráfica, algo especialmente relacionado con las conversaciones con los PNJs.

Todos estos módulos dependerán de un módulo compartido principal que tendrá la tarea de administrar e identificar los recursos disponibles. Los elementos del juego serán representados como una paleta accesible por el usuario en todo momento para seleccionar un gráfico, sonido u otro tipo de recurso soportado por Tennacles.

Para usar esta paleta, Tennacles impone una restricción: toda aventura gráfica dependerá de un directorio, y todos los recursos deberán estar almacenados en este directorio. Una de las principales ventajas de esta restricción es que el usuario no necesitará indicar explícitamente la importación de recursos en su proyecto. Sólo necesita colocarlos dentro del directorio del proyecto usando la organización jerárquica que prefiera, y Tennacles detectará los nuevos elementos o cambios en los ya usados, después de que el usuario seleccione la opción de actualización apropiada (podrá haber varias).

Al usuario no le está permitido manipular o modificar los ficheros generados por Tennacles, los cuales serán colocados en la raíz de la jerarquía de directorios controlada. El resto de los ficheros pueden estar en cualquier parte. Sin embargo siempre podrá ver todos los recursos en cualquier momento. Por supuesto, en el momento que se exporten los datos de Tennacles a un motor específico, el conversor ignorará los recursos no usados por el proyecto, lo que permite tener en el directorio de proyecto más ficheros de los necesarios, útil, por ejemplo, para hacer pruebas.

 <p>Universidad de Deusto Facultad de Ingeniería</p>	<p>1.5 Módulo de personajes y objetos</p>	<p>Proyecto Fin de Carrera</p>
---	---	--------------------------------

El elemento que falta para hacer una aventura jugable es la interfaz del usuario (de ahora en adelante IGU, interfaz gráfica del usuario). La IGU de una aventura gráfica suele estar compuesta por diferentes botones etiquetados localizados en la parte inferior o superior de la pantalla, y los objetos del juego suelen estar representados por gráficos en un inventario virtual. Dado que Tennacles pretende ser una herramienta genérica, no puede especificar la IGU porque cada motor usa una diferente. Por esta razón, la IGU se deja por implementar al plugin exportador.

## 1.5 Módulo de personajes y objetos

Los personajes y objetos de una aventura gráfica no son necesariamente estáticos. Para poder definir su movimiento (ej: el desplazamiento de un personaje), o su secuencia animada (ej: un reloj cuyas manecillas cambian de posición, un televisor del fondo cuya pantalla parpadea, algunas nubes moviéndose en el cielo), el usuario necesita seleccionar un grupo de imágenes, asociarlas con una secuencia y especificar las propiedades de cada imagen (frame de ahora en adelante) de la animación. A través de este módulo el usuario podrá realizar estas tareas y crear sprites animados, que estarán disponibles como elementos individuales desde el gestor de recursos a otros módulos.

Todos los gráficos o sprites usados durante el juego deben tener un “punto caliente” asociado. Este punto caliente será usado como referencia para colocar el gráfico en la escena. Si un gráfico del tamaño de 20x20 pixels tiene su punto caliente en la posición (10, 10), y la posición de pantalla del sprite es (100, 100), el pixel superior izquierdo estará en la posición (90, 90) de la pantalla y el pixel inferior derecho estará la posición (109, 109).

Las animaciones se representan como la ruta que sigue un punto a lo largo del tiempo y el espacio. Este punto estará enlazado con el punto caliente de cada sprite. Para cada sprite se puede indicar su duración y movimiento. La duración será medida en centésimas de segundo. El movimiento indicará el número de pixels que se desplazará el sprite durante ese tiempo especificado. Ambas variables podrán ser introducidas como valores fraccionales: el usuario no verá ninguna diferencia en entre dos sprites que sólo difieren en la parte fraccionaria, pero esta parte no será ignorada a la hora de realizar cálculos en el juego (lo que permitirá por ejemplo desplazar objetos a la velocidad exacta de un pixel y medio por frame en la pantalla).


## 1.6 Módulo de escenas

Este módulo es muy importante dado que los módulos restantes trabajarán normalmente sobre él, usándolo como área de trabajo.

Dado que las aventuras gráficas son representadas en dos dimensiones, para crear un efecto de profundidad el usuario podrá crear varias capas para cada escena, cada una de las cuales tendrá un propósito diferente: la capa de fondo y su Z-buffer, la capa de desplazamiento, la capa de zonas calientes y la capa de personajes y objetos.

### 1.6.1 Capa de fondo

Contendrá los gráficos estáticos que serán mostrados continuamente en pantalla. Sobre éstos se dibujarán los gráficos de los restantes elementos que compondrán la aventura. Para simular el movimiento en los fondos el usuario deberá crear un objeto animado que no tenga interacción alguna con el personaje controlado por el jugador (de ahora en adelante

 <p>Universidad de Deusto Facultad de Ingeniería</p>	<p>1 Documento de objetivos</p>	<p>Proyecto Fin de Carrera</p>
---	---------------------------------	--------------------------------

“ego”) y que sustituirá al fondo. Si el fondo se compone de varios gráficos que sobre salen del área definida como visible, los trozos que superen los límites de la pantalla serán recortados.

### 1.6.2 Capa de Z-buffer

Es un único gráfico de 8 bits de profundidad que indica qué zonas del gráfico de fondo serán dibujadas por encima de los personajes y otros objetos móviles, simulando profundidad en la escena. Los pixels de valor cero indican que el fondo siempre será dibujado por debajo de otros elementos, y un valor de 255 indica lo contrario. Los niveles intermedios indican diferentes niveles o profundidades que indican el orden de dibujado de varios elementos que estén a la vez en el mismo sitio.

### 1.6.3 Capa de desplazamiento

Define las zonas de la pantalla por las que se pueden desplazar los personajes de la aventura. Estas zonas se definirán mediante áreas poligonales, cuyos vértices tendrán un valor porcentual que representa el escalado de los personajes que se desplacen a través de ellas. Por ejemplo, si las esquinas que están a la izquierda de un rectángulo tienen un valor menor que las otras esquinas, eso significará que un sprite que se desplace por esa zona crecerá en tamaño a medida que se desplace de izquierda a derecha. Los valores asignables a cada vértice son porcentajes que representan el grado de escalado que tendrán los sprites. Es decir, si tenemos un rectángulo cuyos vértices tienen el valor de 100, los personajes u objetos cuyo punto caliente se encuentre en esa zona no serán escalados gráficamente, mantendrán su tamaño original.

### 1.6.4 Capa de zonas calientes

Usa también polígonos para especificar qué zonas del gráfico de fondo pueden ser susceptibles de reaccionar ante el cursor del ratón manejado por el usuario durante la reproducción de la aventura gráfica, o pueden modificar el comportamiento de los personajes u objetos que se desplacen a través de ellas.


### 1.6.5 Capa de actores

La última capa permitirá la colocación de personajes y objetos que reaccionan ante la interacción del usuario. Los recursos que sean colocados en esta capa tendrán características como escalado, profundidad en la escena, interacciones de usuario asociadas, condiciones bajo las cuales el recurso aparece en la escena, etc.

## 1.7 Módulo de conversaciones

El usuario final de una aventura gráfica debe ser capaz de hablar a diferentes personajes a lo largo de la partida. Para simular conversaciones, el juego le permitirá escoger al usuario entre diferentes opciones a la hora de hablar. Tras seleccionar una, el PNJ responderá de un modo predeterminado, y si la conversación continua, el usuario será capaz de continuar seleccionando opciones, o quizás entrará también en una conversación predefinida por su parte hasta que de nuevo tenga la opción de escoger una frase del diálogo.



 <p>Universidad de Deusto Facultad de Ingeniería</p>	<p>1.8 Módulo de problemas lógicos</p>	<p>Proyecto Fin de Carrera</p>
---	--	--------------------------------

Las opciones de diálogo mostradas al usuario pueden ser condicionadas por acciones previamente realizadas durante la partida. Por ejemplo, si ego habla con una persona, no tendría mucho sentido que pudiese realizar una y otra vez la misma pregunta: la conversación debería ser coherente, e incluso cuando el contenido de la respuesta debe ser el mismo, algo en la frase debería reflejar que esa respuesta ya ha sido dada (por ejemplo, un personaje podría responder irritado si se le pregunta más de tres veces seguidas algo concreto). Del mismo modo, las opciones mostradas al usuario podrían depender de que ego posea un objeto específico en su inventario, el cual deberá ser recogido en otro escena diferente de la aventura.

Para trabajar con todas las posibilidades, este módulo mostrará las conversaciones como una estructura en forma de árbol, donde cada hilo en la conversación crea una rama. Adicionalmente se podrán especificar condiciones en nodos específicos, que cerrarán o abrirán el paso de la conversación por un camino concreto, una característica estrechamente relacionada con el siguiente módulo (creación y diseño de problemas lógicos), e incluso el nodo raíz puede tener condicionales para alternar el inicio de la conversación con ego. Además, podrá haber nodos especiales que enlazan una parte de la conversación con otra rama, pudiendo crear diálogos cíclicos.

Por esta razón, cada personaje con el que se puede interaccionar tendrá asignado un árbol de conversación, que será representado como un fichero XML salvado dentro del directorio del proyecto, representado por Tennacles como otro recurso cualquiera. Este fichero puede ser modificado por Tennacles, pero dado que este tipo de ficheros consisten únicamente de texto, al usuario se le permitirá modificarlos con cualquier otra herramienta que considere más apropiada para esta tarea. Esto es similar a usar un programa como GIMP [8] o cualquier otro de retoque fotográfico para crear los gráficos del juego. Tennacles no es una herramienta de producción, es más una herramienta de composición que une todas las piezas separadas.


## 1.8 Módulo de problemas lógicos

Este módulo está estrechamente relacionado con el anterior (creación y diseño de conversaciones), ya que controlará toda la activación de condiciones. Aquí, el usuario será capaz de crear y acceder a las variables que desee, las cuales tendrán un valor a lo largo de la aventura. Normalmente booleanas o numéricas, el juego las podrá usar para reaccionar de forma diferente según el valor que contengan. Por ejemplo, el usuario será capaz de especificar que tras recoger un objeto en una escena, la variable “objeto\_necesario\_para\_cruzar\_el\_puente” tendrá un valor de uno. Más tarde, en otra escena, durante la conversación con un personaje, dependiendo del valor de esta variable ego podrá seleccionar un conjunto diferente de respuestas, que llevará a finales de conversación diferentes, creando por lo tanto una sensación de interactividad en el juego.

Las variables pueden ser modificadas por diferentes eventos que podrán ser accionados durante la aventura. Estos eventos podrían ser entrar o salir de una habitación, recoger o soltar un objeto, etc. A cada uno de estos eventos se les podrá asociar la modificación imperativa de una o más variables, afectando a la lógica del juego.

## 1.9 Plugin de exportación Tennacles -> Mad


Una vez que el usuario ha finalizado la tarea de ensamblar los diversos recursos que componen la aventura gráfica, o a mitad del desarrollo de la misma, querrá ver cómo se

 <p>Universidad de Deusto Facultad de Ingeniería</p>	<p>1 Documento de objetivos</p>	<p>Proyecto Fin de Carrera</p>
---	---------------------------------	--------------------------------

visualiza el juego dado que Tennacles no puede ofrecer una visión completa e interactiva de cómo va a funcionar el juego (la herramienta no está orientada hacia ningún motor en particular, y cada uno puede representar el juego de forma diferente). Por esta razón, la opción de exportación invocará un plugin o software externo que cogerá todos los recursos usados por la aventura gráfica y compondrá los datos finales usables por un motor existente. Este proceso es similar a los pasos de compilación en lenguajes de programación como C o C++.

Si hay algún error durante la compilación, se le mostrará al usuario la naturaleza del mismo para que pueda corregirlo (por ejemplo, podría faltar por definir la escena inicial donde comienza la aventura). Si la compilación no tiene errores, una versión final de los datos usables por el motor serán salvados, y al usuario se le preguntará si desea iniciar el motor para ver cómo funciona todo. El motor Mad [17] será usado para este propósito, aunque en el futuro, podría haber diferentes plugins de exportación para cuantos motores existan en ese momento.

Tal y como se ha mencionado antes, la IGU será definida por el plugin, ya que es algo muy específico de cada motor. Hay motores tradicionales que muestran en la parte inferior de la pantalla un menú de acciones junto con un inventario de objetos de tamaño fijo. Otros motores muestran el inventario como una pantalla emergente donde todos los objetos son representados por iconos, a veces con descripciones. Esta última opción es un patrón habitual en los juegos comerciales dado que no ocupa espacio de pantalla, el cual puede ser aprovechado para mostrar más gráficos, y así la interfaz puede ser contextual/automática. Este tipo de interfaces no pueden ser creadas de forma genérica: lo mejor es que haya una específica para cada motor, y es por esa razón por la que la IGU será seleccionada o definida desde el plugin exportador.

 <p>Universidad de Deusto • • • • Facultad de Ingeniería</p>	2 Justificación del proyecto	Proyecto Fin de Carrera
---	------------------------------	----------------------------

## CAPÍTULO 2

# Justificación del proyecto

### 2.1 Videojuegos

---

#### 2.1.1 Evolución


El carácter lúdico siempre ha estado presente en el desarrollo de los ordenadores y la informática. Mucho antes de dominar complejas tablas en bases de datos, soportar múltiples usuarios de forma concurrente o transferir grandes cantidades de datos de un lado a otro del mundo, los videojuegos han existido en formas a veces tan simples como el juego de adivinar un número dentro de un rango.

Quizás el comienzo más claro de los videojuegos fue cuando dos rayas y un punto (Pong) pegaron a muchos aficionados a sus pantallas. Este hecho marcó el comienzo de una carrera desenfrenada, primero por averiguar cómo era posible hacer que máquina proporcionase entretenimiento, y posteriormente por conseguir crear el mejor entretenimiento posible.

Hacia falta tan poco para conseguir hacer un juego, y sin embargo, a día de hoy quizás el software que más tire de los diseñadores de hardware sea el lúdico, a medida que los usuarios demandan experiencias cada vez más intensas, con gráficos más realistas y efectos sonoros capaces de emular atronadoras tormentas.

Fue entre la década de los 80 y cuando hubo un estallido de producción de videojuegos para máquinas de 8 bits, baratas, sencillas de usar, y capaces de entretener a cualquier aficionado de la época. Dadas las limitaciones del hardware, los programadores capaces de aventurarse por los entresijos de las CPUs se bastaban para producir juegos. Ya era difícil hacer que aquellos “trastos” resultasen entretenidos, así que nadie exigía complejos efectos gráficos o simuladores tridimensionales realistas.

Pero con la llegada de los 16 bits comenzaron a proliferar máquinas que dibujaban más de 16 tristes colores en pantalla, rotar dibujos o reproducir sonidos increíbles, y lo que antes era una tarea de una persona comenzó a necesitar de alguien con aptitudes gráficas. No es difícil dibujar el gráfico de un personaje cuando sólo se disponen cuatro colores y 16x16 pixels de pantalla. La cosa cambia cuando se pueden mostrar imágenes fotorealistas y para hacerlos hay que pasar muchas horas dibujando una única imagen.

 <p>Universidad de Deusto Facultad de Ingeniería</p>	<p>2 Justificación del proyecto</p>	<p>Proyecto Fin de Carrera</p>
---	-------------------------------------	--------------------------------

En la década de los 90 ya comenzaban a formarse grupos de programadores y algún artista o músico que formaban compañías para hacerse multimillonarios cumpliendo las fantasías de los millones de compradores repartidos por todo el mundo. Lo que habitualmente comenzaba como un hobby personal se convirtió por fin en un negocio lucrativo, y como tal, era necesario ofrecer siempre un mejor producto que la competencia. Fue en este tiempo cuando las empresas ya formadas contrataban artistas de forma prolongada, ya que antes sólo eran contratados cuando el juego se sabía que iba a funcionar y estaba casi totalmente programado.

Cuando un grupo excede la cantidad de una persona, se deja de dedicar el 100% del tiempo al desarrollo y cuantas más personas hay, más se dedica a la gestión y sincronización de personal. Además, se comenzó a tratar los videojuegos como productos de consumo masivo al igual que las películas o la música, y las empresas dedicaban cada vez tiempo más al marketing dando a conocer sus productos en las revistas especializadas. El videojuego, al menos a nivel comercial dejó de ser una empresa de una o dos personas.

### 2.1.2 Hechos actuales


Hoy en día entrados en el siglo XXI, con anuncios de videojuegos hasta en los periódicos o la televisión, se entiende la producción de un juego comercial como una empresa económica de alto riesgo con una gran inversión de capital durante al menos uno o dos años para que un equipo de a partir de una treintena de personas pueda lanzar su producto al mercado y esperar que la inversión en tiempo, trabajo y dinero haya merecido la pena y puedan obtener beneficio alguno.

Algunos programadores ya son conocidos como auténticas estrellas del rock con sueldos espectaculares y millones de fieles seguidores: John Carmack (saga “Doom” y “Quake”), Peter Molineux (desde “Populous” hasta “Black and white”), Chris Roberts (saga “Wing Commander”), Lord Richard Garriott (saga “Ultima”), y así muchos otros. Si bien ellos son recordados por crear los juegos más originales o escribir los motores gráficos técnicamente más avanzados, todos ellos son parte de grupos que a veces alcanzan y superan la cifra de 100 personas.

Lo más sorprendente es que las proporciones estén cambiando. Hace 20 años todos los integrantes del grupo eran programadores, que si no tenían a mano alguien con habilidades artísticas tenían que suplirlo ellos mismos. El programador tenía la idea, fabricaba el soporte necesario (software) que la pudiese representar, y finalmente dibujaba un par de garabatos que se movían a trompicones por la pantalla.

Ahora el grupo de programadores no suelen ser los que dan la idea, que viene casi al completo depurada por diseñadores especializados, y tampoco son los encargados de dibujar los gráficos, que son realizados por artistas de diferentes categorías y especialidades. Los programadores ya sólo son responsables de dar soporte al resto del grupo de personas que trabajan con ellos.

Si por ejemplo tomamos los créditos de una reciente producción de la empresa Blizzard [2], de unas 300 personas involucradas en el desarrollo de “Warcraft III: Reign of chaos”, un popular juego ambientado en el ambiente fantástico medieval de orcos, humanos y elfos, 33 estuvieron a cargo de tareas relacionadas con la programación, mientras que el resto se repartían en artistas, diseñadores, escritores, productores, técnicos de la industria del cine, músicos, actores, y un largo etcétera. Quizás sea injusto calcular así el porcentaje de programación necesaria para desarrollar un videojuego, pero según esas cifras, sólo un 10% de los recursos humanos usados en la producción de un juego de hoy en día son

 <p>Universidad de Deusto Facultad de Ingeniería</p>	<h2>2.1 Videojuegos</h2>	<p>Proyecto Fin de Carrera</p>
---	--------------------------	--------------------------------

dedicados a la programación.

Este hecho se ve corroborado por la escena de programadores amateur. Gracias a librerías como Allegro, SDL [30] o DirectX [5] de Microsoft, se puede decir que cualquiera tiene la posibilidad de programar un juego. Esto se ve reflejado en la cantidad de programas gratuitos realizados por programadores amateur o profesionales realizados en su tiempo libre. Pero lo que también se ve es que muchos de estos juegos, algunos impecables a nivel técnico, no se sostienen de cara a un usuario que en lugar de pedir calidad técnica busca gráficos superlativos, argumentos originales, jugabilidad extrema y adicción, razón por la cual no suelen gozar de un gran público en relación con los juegos comerciales.

Aquí es donde entra en juego un proyecto como Tennacles. Tennacles asume directamente que la parte de programación está completamente superada. Ahora bien, ¿cómo puede alguien reunir el talento de varias personas y combinarlo para crear un videojuego adictivo?

### 2.1.3 El género de las aventuras


Dentro del inmenso mundo de los videojuegos existen varias categorías y muchas zonas grises que se suelen clasificar como híbridas. Son conocidas las categorías de acción, deportes o plataformas. El género que nos ocupa, el de las aventuras, quizás fue uno de los primeros en ser de carácter popular, por atraer la mayor cantidad de usuarios de todos tipos y edades.

Las aventuras se suelen caracterizar por ser juegos de tipo pausado en los que antes que el músculo hay que usar la inteligencia y tener buenas dotes de paciencia y contemplación para poder continuar avanzando en la historia. Este último punto es también singular: a diferencia de los demás géneros, en los cuales no suele ser necesario un argumento, y cuando lo hay, únicamente existe para justificar la pulsación descontrolada de las teclas del ordenador, en una aventura lo que más atrae es el argumento, la historia que se desenvuelve en torno al protagonista. Se crea así un mundo mágico que envuelve al usuario para que se sienta como el investigador que descubre al perpetrador de un crimen, un rey destronado que busca justicia tras ser exiliado, o un héroe que busca la estrategia correcta para deshacerse de unos malvados alienígenas que nos atosigan a los indefensos habitantes de algún planeta lejano.

Aun cuando todavía los ordenadores apenas poseían capacidades de representación gráfica, eran los argumentos los que hacían furor con las aventuras conversacionales. “Zork” fue uno de las primeras aventuras, creada en 1977 por Marc Blank, Bruce K. Daniels, Tim Anderson, y Dave Lebling. Sólo se leía texto y se interactuaba mediante comandos, que además debían ser escritos de forma exacta a como los autores los habían codificado. No obstante este juego causó furor, y fue en 1980 cuando se vendió la primera copia para la máquina PDP-11. Se desató la “zorkmanía” y más tarde llegarían secuelas y otras empresas competidoras que con sus productos aumentarían la popularidad de este reciente género.

Cuando llegaron los primeros ordenadores con capacidades gráficas se anunció de forma silenciosa el comienzo del fin de las aventuras conversacionales, pues éstas serían reemplazadas por las aventuras gráficas. De la mano de “On-Line Systems”, que pasaría a llamarse “Sierra On-Line” [31] en 1982, llegaron las primeras aventuras para el Apple donde aparte del sobrio texto se mostraban los primeros gráficos que intentaban representar las estancias por las que paseaba el protagonista.

En cuanto los gráficos mejoraron para dejar de ser garabatos, las aventuras gráficas

 <p>Universidad de Deusto Facultad de Ingeniería</p>	<p>2 Justificación del proyecto</p>	<p>Proyecto Fin de Carrera</p>
---	-------------------------------------	--------------------------------

se convirtieron en clásicos, como la saga “King quest”, la saga “Space quest”, “Maniac Mansion”, la saga “Monkey Island” o la saga “Indiana Jones”. Pronto se liberaron de la herencia textual usada para describir las escenas y adoptaron un formato gráfico donde la mayor parte de la pantalla se dedicaba a representar la escena actual, con una zona inferior o superior para mostrar las acciones que puede realizar el protagonista o el inventario.

Precisamente una de las aventuras gráficas que elevaron el listón por su impresionante calidad fue “The day of the tentacle”, una producción de Lucasarts en 1993. De ahí surgió el título del proyecto, que responde a una degeneración fonética de la palabra “tentacle”. El máximo logro al que puede aspirar Tennacles es ser lo suficientemente útil como para que alguien pueda crear algún día una aventura de la calidad de “The day of the tentacle” usando esta herramienta.

En su última fase evolutiva, las aventuras gráficas decidieron tomar por completo la pantalla para representar el escenario (“Full Throttle”, “The Dig”), a veces pasándose incluso a tecnologías tridimensionales (“Grim Fandango”). En estos juegos la interfaz se simplificó a un par de acciones básicas, como andar, usar objetos y hablar con los personajes, para poner más énfasis en los gráficos, los diálogos (que ahora incluso eran recitados por actores) y las secuencias cinemáticas.

Indiferentemente de la época, tecnología o estilo, las aventuras siempre han tenido tres elementos en común: escenas, objetos y personajes. Las escenas son siempre el marco donde se desarrolla la acción, y en ellas existen los objetos y los personajes. Los objetos son siempre elementos que se tienen que recoger en una escena y usarlos o combinarlos con otros en otra escena. Conseguir encontrar la combinación adecuada suele ser el puzzle que una vez resuelto nos desvela más escenas en el juego, pudiendo evolucionar en la trama.


Los personajes son “objetos especiales”. Tradicionalmente no tienen movilidad y suelen estar fijos en una escena. No los podemos añadir al inventario (excepto en aventuras gráficas de fantasía o humor desenfadado), y suelen limitarse a la interacción mediante objetos o diálogo. Los puzzles son por lo tanto encontrar un objeto que necesite un personaje, o dialogar con el personaje hasta obtener la información necesaria para que sepamos cómo se resuelve otro puzzle.

Es en los diálogos donde se inserta la interactividad perdida de las aventuras conversacionales. En aquellas, el usuario tecleaba comandos similares a “PREGUNTAR POR MARÍA” o “PREGUNTAR POR MESA”, y estas preguntas desencadenaban las respuestas necesarias para que podamos avanzar. Hoy en día, el diálogo se suele interrumpir y se nos muestran varias opciones de las cuales debemos elegir una, la cual será usada por el protagonista. Este método quizás reste flexibilidad, pero permite a los diseñadores mayor creatividad, pues si en las aventuras conversacionales no se nos ocurría un determinado tipo de pregunta, era imposible continuar jugando.

### 2.1.4 Objetivo de Tennacles

La técnica necesaria para representar una aventura gráfica está por completo desarrollada e implementada en varios motores, algunos de los cuales incluso son gratuitos. Pero muchos de estos motores todavía no están orientados a un usuario final que no sepa programar, limitando su uso.

Tennacles tiene por objetivo facilitar el desarrollo de una aventura gráfica al simplificar el proceso de composición de las escenas, que consiste en unir los gráficos, sonidos y diálogos. Así, con esta herramienta, y partiendo de unos recursos ya existentes, se permitirá diseñar mediante una interfaz gráfica una aventura que pueda ser reproducida en un

 <p>Universidad de Deusto Facultad de Ingeniería</p>	<h2>2.2 Python</h2>	<p>Proyecto Fin de Carrera</p>
---	---------------------	--------------------------------

motor existente. Las herramientas de características similares a Tennacles no llegan a una decena,

Los últimos juegos como “Black and white” o “The Sims” comienzan a ser una especie de “meta-juegos”, ya que en ambos el objetivo es asumir el papel de una entidad del juego superior a la mayoría de los personajes (similar a convertirse en una deidad) para proporcionarles a estos un entorno para vivir o relacionarse. Quién sabe, quizás dentro de unos años los programas como Tennacles sean habituales, y el mayor furor no sea jugar a juegos, sino diseñarlos.

## 2.2 Python

### 2.2.1 ¿Qué es?

Python es un lenguaje de programación de muy alto nivel excepcional en claridad y simplicidad de expresión. Python permite a los programadores construir las soluciones a sus problemas usando modos humanos de pensamiento y lógica. Aunque Python es relativamente simple de aprender y usar, su extensiva librería estándar, portabilidad, y habilidad para integrarse estrechamente con otros lenguajes lo hacen una herramienta poderosa para muchos tipos de desarrollo de software.

Python fue creado por Guido van Rossum en 1989 como un método de enseñar a gente inexperta los conceptos más sofisticados y poderosos de la programación de ordenadores. Desde entonces ha crecido hasta convertirse en una plataforma usada por muchas empresas a lo largo y ancho del planeta. Python es un lenguaje de programación orientado a objetos, dinámico, interpretado y multiplataforma implementado en C.

La mezcla de características de Python es incomparable por cualquier otro lenguaje de programación. Combina flexibilidad con gracia, lógica con claridad, franqueza con energía, la velocidad del desarrollo con capacidad de mantenimiento, y la portabilidad con escalabilidad. La simplicidad, el poder, y la portabilidad lo hacen ideal para una amplia gama de aplicaciones - desde pequeños programas a sistemas grandes y sofisticados. La claridad de la expresión lo hace igualmente útil para la especificación y prototipado del software, donde actúa como “pseudo-código ejecutable”.

Python es un lenguaje híbrido que mezcla los paradigmas de la programación procedimental, orientada a objetos, y funcional, ofreciendo a los programadores la opción para utilizar la herramienta correcta para construir cada parte de su solución.

Python funciona en casi todas las plataformas de hardware, desde PDAs hasta Mainframes. Soporta todos los sistemas operativos importantes incluyendo casi todas las variantes de Unix, Windows, y Mac OS. Los programas escritos puramente en Python realmente cumplen la promesa de escribir una vez, ejecutar donde sea.

Como lenguaje interpretado, Python ahorra tiempo de desarrollo; elimina la compilación del ciclo de la codificación y depuración. Python se integra fácilmente con C, C++, y Java, los cuales pueden ser usados para escribir las porciones críticas en velocidad de la aplicación, sin tener que sacrificar las ventajas de Python para el desarrollo del resto del sistema. La capacidad de interoperar con otras idiomas también lo hace ideal para integrar antiguas aplicaciones heredadas.

Pero Python no es sólo un lenguaje de programación; es una plataforma completa del desarrollo con una biblioteca estándar extensa y un número enorme de extensiones disponibles que tratan virtualmente cualquier necesidad de programación. Numerosas opciones



están disponibles para la interfaz, la web, servicios cliente/servidor, bases de datos, CORBA, proceso numérico, estadístico, de imagen, y muchos otros tipos de programación.

Todos estos factores combinados hacen de Python el lenguaje idóneo para implementar casi cualquier sistema.

### 2.2.2 Toma inicial de contacto


La primera característica que llama la atención es que Python no usa delimitadores de bloques de código como hacen otros lenguajes más populares, y se usa el tabulado o sangrado del código para indicar al intérprete los bloques de código. Es difícil explicar esta característica, así que a continuación se muestra la función de Fibonacci codificada en C y en Python para que sirva de comparativa:

<code>#include &lt;stdio.h&gt;</code>	00	<code>#!/usr/bin/env python</code>
<code>/* Imprime la serie de Fibonacci hasta n */</code>		<code># Imprime la serie de Fibonacci hasta n</code>
<code>void fib(int n)</code>		<code>def fib(n):</code>
<code>{</code>	05	<code>    a, b = 0, 1</code>
<code>    int a = 0, b = 1, t;</code>		<code>    while b &lt; n:</code>
<code>    while (b &lt; n) {</code>		<code>        print b,</code>
<code>        printf("%d ", b);</code>		<code>        a, b = b, a+b</code>
<code>        t = b;</code>		
<code>        b = a + b;</code>	10	<code>if __name__ == "__main__":</code>
<code>        a = t;</code>		<code>    fib(2000)</code>
<code>    }</code>		
<code>}</code>		
<code>int main(void)</code>	15	
<code>{</code>		
<code>    fib(2000);</code>		
<code>    return 0;</code>		
<code>}</code>	19	

La ausencia de llaves en el código de Python es lo más llamativo. ¿Dónde acaba la definición de la función o cuándo termina el bucle? El número de espacios entre el margen izquierdo y la primera letra del código delimitan el bloque. Lo que parece una inflexibilidad se transforma tras unos minutos de práctica en un agradecimiento:

1. Se teclean menos caracteres para el mismo código y se evitan líneas dedicadas en exclusiva a delimitadores para “embellecer” el código.
2. Debido a que en C las llaves se pueden poner en cualquier sitio, incluso todo el código se puede escribir en una línea, en el primer momento en el que se lee código no escrito por nosotros hay que esforzarse para saber dónde comienzan y acaban los bloques, cada programador tiene su estilo. En Python es obligatorio marcarlos con el nivel de sangrado, así que sólo hay una manera de escribir un bucle: de la forma correcta.
3. Se obtiene mayor claridad, la lectura escalonada del código de Python resulta más sencilla y natural.



 <p>Universidad de Deusto Facultad de Ingeniería</p>	<h2>2.2 Python</h2>	<p>Proyecto Fin de Carrera</p>
---	---------------------	--------------------------------

Por supuesto, esto no es ni mucho menos el comienzo de las ventajas de Python, pero ya aquí se ve que una de las filosofías de Python: para programar algo, sólo debería haber una manera, y ésta debe ser obvia. En la práctica esto se traduce a que el código de Python sin apenas modificaciones podría pasar por el pseudo-código usado en algunas clases teóricas dedicadas a explicar algoritmos. Al olvidar complejas reglas de sintaxis que nos limitan, uno puede expresar sus ideas de forma más libre.

Otros detalles comunes a la mayoría de los lenguajes interpretados son el hecho de que no hace falta declarar las variables. En Python, este proceso es automático, si bien hay una pequeña diferencia: nunca se declaran variables. Cuando el intérprete procesa la línea ‘a, b = 0, 1’, lo que hace es crear un los nombres ‘a’ y ‘b’ en el entorno, y estos apuntan a los objetos 0 y 1 respectivamente.

En Python todo es un objeto, incluso las propias funciones lo son, pudiendo tener atributos como las clases de C++ o Java y pudiendo crear meta-funciones que crean funciones. En otros lenguajes cuando se declara una variable el programador sabe que está reservando una determinada cantidad de memoria e intentar escribir fuera de esa memoria causa los típicos errores de desbordamiento de buffers. En cambio aquí el programador maneja nombres, que en realidad son punteros o referencias a los elementos (de hecho, no existen conceptos como el paso de parámetros por copia, naturales en C). Por esta simple razón Python “no soporta” desbordamientos de buffer, el programador no tiene acceso físico a la memoria.

Al igual que Java, Python tiene su propio recolector de basura, evitando al usuario la gestión manual de la memoria. La única posibilidad de tener problemas con la gestión de memoria es mantener siempre referencias “vivas” a objetos para evitar que el recolector de basuras pueda destruirlos. El concepto de referencia “viva” es similar al de la existencia de una variable en un bloque de código en C. En el ejemplo anterior, tanto en C como en Python las variables ‘a’ y ‘b’ sólo existen dentro del ámbito de la función. En cuanto el intérprete termina la ejecución de la función, las variables ‘a’ y ‘b’ pasan a estar “muertas” porque ya nadie hace referencia a ellas, y entonces el recolector puede reusar su memoria.

Otra característica común con Java son las excepciones, lo que permite realizar un tratamiento de errores superior al que se puede hacer en C. Una diferencia respecto a Java es que Python permite implementar la herencia múltiple, emulada en Java a través de las interfaces. No sólo es así, sino que en Python las funciones que reciben clases no verifican el tipo de la clase que reciben como parámetro, únicamente invocan los métodos que necesitan. Si el objeto ha implementado esos métodos, todo funcionará perfectamente. Se lanzará una excepción si hay algún problema, por supuesto.

### 2.2.3 Tipos de datos

Una forma de evaluar la potencia de un lenguaje es analizar los tipos de datos nativos que soporta. Los tipos de datos proporcionados por la STL en C++ no son nativos, ya que son externos al lenguaje original. Al margen de los tipos básicos como enteros, flotantes o cadenas de texto, Python tiene listas, tuplas y diccionarios como los tipos de datos nativos más habituales y más usados. Hay otros como los números complejos o enteros de longitud indeterminada, pero no suelen ser tan útiles como los que se describen a continuación.

#### ■ Listas

Similares a los vectores de Java, las listas son objetos que tienen unas cuantos métodos y actúan como contenedores de otros elementos. A diferencia de Java, una lista puede



contener elementos heterogéneos: el primer elemento puede ser un entero, el siguiente una cadena, el tercero una clase definida por el usuario, etc. En Java esto es posible perdiendo el tipo original del dato, ya que éste es convertido a un objeto genérico, lo cual imposibilita al receptor de un vector saber qué tipo original tenían los elementos originales, reduciendo así la facilidad de implementar el polimorfismo.

En Python además hay tres herramientas funcionales que trabajan con listas: filter, map y reduce. map aplica una función a cada elemento de la lista y devuelve una nueva lista con el resultado de la llamada a esta función. filter aplica una función a cada elemento de la lista. El resultado de la función es tratado como un valor booleano, y la lista de elementos que devuelve contendrá solamente aquellos que devolvieron el valor verdadero. Finalmente reduce aplica una función binaria sobre los dos primeros elementos y los sustituye por el resultado, consumiendo en el proceso una lista y devolviendo un único valor final.

Las listas se usan también como pilas o colas. Dado que son objetos normales y corrientes, tienen métodos, y los tipos de datos descritos se pueden implementar usando los métodos 'push', 'pop', 'insert' y 'append'. Entre otros métodos útiles está el 'sort', que permite ordenar listas comparando los objetos que hay en ellas.

## ■ Tuplas

Las tuplas son similares a las listas, aunque tienen la particularidad de ser inmutables. ¿Qué es esto? En una lista se puede modificar el tercer elemento por otro. Esto no pasa con las tuplas. Las tuplas se suelen usar de forma implícita y transparente al programador, a diferencia de las listas. Se ve el uso de una tupla en el ejemplo anterior durante la asignación de los valores 'a' y 'b'.


La coma entre los elementos a ambos lados de la asignación implica la creación temporal y transparente de una tupla. Lo que se consigue así es el empaquetado y desempaqueado de tuplas de forma automática: primero los valores de la derecha de la asignación se empaquetan en una tupla, y ésta al asignarse a los valores de la izquierda se desempaqueta, creando la ilusión de haber realizado una asignación múltiple. El beneficio natural más común es que el programador implementa de forma natural funciones que devuelven más de un elemento a la vez, olvidando la necesidad de parámetros específicos de entrada y salida.

Dicho sea de paso, las cadenas en Python también son inmutables.

## ■ Diccionarios

Este tipo de dato es también conocido como un array asociativo. Se trata pues de un array cuyos índices no son sólo números, sino que pueden ser objetos. Los diccionarios están implementados de forma muy eficiente, y a menudo se usan en algoritmos que deben encontrar duplicados de objetos o extraer valores de un conjunto según unos índices aleatorios.

Un diccionario consiste entonces de la pareja llave/valor, y puede haber en un diccionario tantos elementos como llaves diferentes haya. Es posible recorrer el contenido de un diccionario según sus llaves, valores o ambos simultáneamente (gracias al uso transparente de las tuplas anteriormente descritas).

 <p>Universidad de Deusto Facultad de Ingeniería</p>	<h2>2.2 Python</h2>	<p>Proyecto Fin de Carrera</p>
--	---------------------	--------------------------------

### 2.2.4 Características avanzadas

Python destaca sobre lenguajes como C, C++ y Java porque es totalmente dinámico y no hace comprobaciones de tipos. Pero a esto hay que añadirle algunas herramientas funcionales traídas de otros lenguajes como LISP [22] o Haskell [16]. Esto extiende el rango de acción del lenguaje, en lugar estar obligado a solucionar todos los problemas mediante las estrategias habituales, a veces se puede optar por soluciones más exóticas y a menudo mejores en términos de rendimiento y legibilidad. Hay muchas más características en el lenguaje que las aquí descritas, pero estas son a mi juicio las que desmarcan a Python de cualquier otro competidor.

#### ■ Funciones anónimas

Las funciones anónimas se crean con la palabra clave ‘lambda’. Una función anónima es creada y usada en el mismo momento donde ésta es declarada. Sería posible crear una función anónima mediante ‘lambda’ y asignarla a un nombre, con lo cual se obtendría una función normal y corriente. Pero el objetivo de las funciones anónimas consiste en definir pequeñas funciones que realicen alguna tarea y que como sólo se van a usar en un sitio y son específicas, escribir su código completo en otro lugar resulta incómodo para el programador. Las funciones anónimas se suelen usar casi siempre con una de las tres herramientas funcionales anteriormente descritas. Ejemplo:

```
lista = filter(lambda x: x > 5, [1, 8, 19, 3, 7])
```

En este ejemplo tenemos una lista con cinco elementos enteros. Tal y como se indicó anteriormente, la función filter aplica una función sobre cada elemento de una lista y devuelve una lista nueva que contiene sólo aquellos que devolvieron un valor de verdadero. Aquí, la variable ‘lista’ acabaría siendo una lista con los elementos 8, 19 y 7. Además, las funciones anónimas pueden tener varios parámetros:

```
valor = reduce(lambda x, y: x * y, [2, 3, 4, 5])
```

En este ejemplo vemos otra herramienta funcional, la función ‘reduce’, que aplica una función binaria a cada elemento de una lista. El resultado vuelve a ser usado con el siguiente valor de la lista, y así hasta consumirla entera. Como se puede esperar, la variable valor será el resultado de ‘ $2 * 3 * 4 * 5 = 120$ ’.

En los lenguajes que no tienen funciones anónimas, todas deben ser descritas en alguna parte, lo que puede ser un incordio para este tipo de funciones minúsculas dado que hay más código que define la función que la propia función. En la sección dedicada a PyGTK se verá un ejemplo aplicado al proyecto que realzará la utilidad de las funciones anónimas de cara al programador.

#### ■ Todo es una interfaz

En Python cuando se programa una función, nunca se escribe código que “pregunta” por el tipo de objeto para ejecutar un trozo de código u otro. En lugar de eso, se declara en la documentación que la función recibe objetos que implementen un grupo determinado de métodos o firmas. En la práctica significa que una función puede aceptar como



parámetro objetos que no estén relacionados por herencia y ejecutar un método que se llama igual en ambos (polimorfismo).

Se puede entender, de hecho, que la función programada requiere una interfaz que esté implementada por los objetos que procesa. Como el lenguaje es dinámico, a diferencia de Java no hace falta declarar en ninguna parte las interfaces que implementa un objeto, simplemente se pasa a la función y si no soporta el método la función lanzará una excepción para indicar que el objeto no implementa el método que se quería invocar. Las ventajas de esto es que un programador sólo debe cumplir el mínimo posible para que una función pueda trabajar sobre una clase del usuario.

El resultado de este tipo de interfaces serían las funciones que manejan objetos de tipo fichero para escribir o leer en ellos. El usuario puede crear entonces una clase e implementar los métodos que usa la función, que suelen ser 'read()' y 'write()' o sus paralelas 'readline()' y 'writeline()'. Podría entonces crearse un objeto que hace que una función escribe en un fichero, pero en realidad el objeto manda el texto por red a un ordenador remoto. Esto incrementa la reusabilidad del código, ya que puede funcionar en situaciones que no estaban previstas por los autores.

## ■ Iteradores y generadores

Los iteradores son de sobra conocidos, pero lo que muchos no conocen es la transparencia de éstos en Python. Los iteradores son tan transparentes que de hecho el programador usuario no los maneja de forma directa porque están bajo una interfaz. Un uso posible de los iteradores es hacer que un tipo de datos se pueda recorrer como otro tipo de datos diferentes. El ejemplo por excelencia es el bucle que recorre los nodos de un árbol. Pero veamos primero por qué un programador usuario no ve los iteradores:

```
for x in [1, 2, "Juan", 8.9]: print x
```

En este ejemplo recorreremos una lista imprimiendo los elementos por pantalla. Podríamos pensar que estamos haciendo un bucle como el 'for' de C, pero no es así. En realidad Python construye un iterador para la lista indicada, y en cada interacción llama de forma automática el método 'next()' del iterador para que devuelva el siguiente elemento de la secuencia. Veamos otro ejemplo práctico:

```
file = open("fichero.txt", "rt")  
for line in file: print line  
file.close()
```

Aquí se ve otro bucle, pero no se realiza a lo largo de una lista. El objeto 'file' de tipo fichero implementa el iterador. La instrucción 'for' detecta este hecho, y permite iterar a través del fichero. En este caso, el iterador llama a la función 'readline()' del objeto que devuelve las líneas del fichero como cadenas de texto. En definitiva, el código de ejemplo muestra por pantalla el contenido del fichero.

Pero lo más asombroso es ver la forma de implementar un generador que soporta el protocolo de iteración. En primer lugar, hay que ver cómo funcionan las llamadas a función de Python o C. Cuando se llama a una función, ésta crea su propio espacio de trabajo donde se crean sus variables locales (en C esto equivale a la pila). Cuando la función llega a su fin o a la palabra clave que marca su retorno, las variables locales son destruidas y



el valor de la función (si tiene alguno) es devuelto al llamador. Una llamada posterior a la misma función creará otro nuevo conjunto de variables locales frescas y sin usar. Pero, ¿y si las variables locales no fuesen destruidas al terminar la función? Esto es lo que es un generador, una función que se puede continuar. Aquí hay un ejemplo del código de Tennacles:

```
def walk(path, strip_length = 0):
    if not strip_length:
        strip_length = len(os.path.join(path, ".")) - 1

    for item in os.listdir(path):
        full = os.path.join(path, item)
        if os.path.isdir(full):
            for val, partial in walk(full, strip_length):
                yield val, partial
        else:
            yield full, full[strip_length:]
```

Este ejemplo más complejo que los demás recorre todo el árbol de ficheros a partir de la rama 'path' indicada. En lugar de devolver una lista completa con todos los nombres del fichero, lo cual consumiría mucha memoria, la palabra clave 'yield' funciona como un retorno especial.

En realidad, la función devuelve un objeto generador (similar al que estaba implementado en el fichero del ejemplo anterior) que soporta el protocolo del iterador. Cuando el intérprete ejecuta la palabra clave 'yield', el generador devuelve el valor (en este caso una tupla de dos elementos). La gran diferencia entre el comando 'yield' y 'return', es que 'yield' suspende el estado de ejecución del generador y las variables locales son preservadas.

En la siguiente llamada al método 'next()' del generador, la función continuará la ejecución inmediatamente después de la palabra clave 'yield'. Dicho de otro modo, se consigue asociar estado a una función. El código que usase este generador quedaría así de simple:

```
for ruta, nombre_directorio in walk("."):
    print ruta, nombre_directorio
```

Como se puede ver, de cara a un usuario programador el generador parece que devuelve una lista completa que es recorrida por el bucle. A nivel interno esta lista nunca existe y el uso de la memoria es optimizado al máximo pues sólo se consume la memoria de un elemento de toda la lista. De cara al usuario implementador, el generador es una forma sencilla de preservar el estado entre una y otra llamada consecutivas a su función. Evita el engorro de obligar al usuario a que pase manualmente un parámetro que almacene el estado de la función y que deba tratar con su iniciación y destrucción, tal y como ocurre con las funciones C 'opendir', 'closedir' y 'readdir' del estándar POSIX.

### ■ Acceso controlado a los atributos

La metodología tradicional respecto al diseño de objetos que deben exponer atributos consiste en declararlos protegidos o privados para que el usuario no pueda acceder directamente a ellos e implementar un conjunto de funciones (que tradicionalmente usan



el nombre del atributo con el prefijo ‘get’ o ‘set’) para proporcionar esta funcionalidad básica.

La ventaja de este “enmascaramiento” consiste en que una vez declarada la interfaz para modificar o recuperar los atributos, el código que llama a estos métodos no tiene que cambiar aunque lo haga la implementación del objeto. Pero esto se lleva al extremo en una errónea previsión de que todo puede cambiar. Además, si todo consiste en asignar variables u obtener su valor, ¿no sería fantástico que pudiésemos hacerlo como si siguiesen siendo variables?

Precisamente esto es lo que permite hacer Python con las propiedades. Consiste en crear una variable virtual que tiene asociados tres métodos: uno para obtener el valor, otro para asignar un valor, y otro para borrar la variable (habitualmente será una función vacía para que el usuario obtenga un error al intentar hacerlo). Aquí se puede ver un ejemplo que define una propiedad y posteriormente el código que la usa:

```
class C(object):
    def get_tamano(self):
        resultado = ... computación ...
        return resultado

    def set_tamano(self, tamano):
        ... calcular algo basado en la variable
        tamano y el estado interno del objeto ...


    # Define una propiedad sin función de borrar
    tamano = property(get_tamano, set_tamano,
                     None, "Tamaño del objeto")
    ...
obj = C()          # crea un objeto de la clase C
obj.tamano = 50   # asigna un tamaño de 50 unidades
print obj.tamano  # imprime por pantalla el tamaño
```

¿Para qué podría ser esto útil? Pues por ejemplo para implementar el patrón “observer”. El usuario sólo ve una asignación a un atributo del objeto, pero por debajo esa asignación podría estar notificando las interfaces de la aplicación que muestran una vista del tamaño del objeto. Igualmente resulta esto útil en el encapsulamiento de otras clases que por una u otra razón no queramos que el usuario vea, o simplemente para simplificar el código que debe escribir el usuario programador. Finalmente, podemos crear propiedades con algunos métodos a ‘None’. En el ejemplo se crea un atributo que no se puede borrar. Podríamos hacer uno que no se pudiese escribir, creando atributos de sólo lectura.

## ■ Gran librería estándar

Dado que Python comenzó su andadura en 1989, ha habido suficiente tiempo como para que toda una comunidad de desarrolladores extienda el lenguaje. Al igual que en Java, Python permite crear paquetes que agrupan clases y funciones. Otra filosofía de Python es que “viene con pilas incluidas”, lo que se traduce en que cada nueva versión del lenguaje ha añadido a su librería estándar de paquetes las mejores extensiones desarrolladas por la comunidad del software libre.

Tras más de 10 años de desarrollo, simplemente es difícil encontrar áreas de la informática para las que no se hayan escrito módulos de Python para facilitar el trabajo. Así, la

 <p>Universidad de Deusto Facultad de Ingeniería</p>	<h2>2.2 Python</h2>	<p>Proyecto Fin de Carrera</p>
---	---------------------	--------------------------------

distribución reciente incluye paquetes para trabajar con XML, paquetes de computación científica (cálculo numérico, simbólico, dibujado de gráficos de alta calidad, etc), interfaz gráfica, acceso estandarizado a bases de datos libres o comerciales, programación de páginas web dinámicas o tareas relacionadas con el procesamiento de páginas web, soporte de aplicaciones multi hilo, acceso directo a sockets o módulos de alto nivel que encapsulan los protocolos POP, IMAP, SMTP, HTTP, y un largo etcétera.

### 2.2.5 Otras opiniones sobre Python

Python todavía no es un lenguaje de masas como lo son C++ y Java. Pero difícilmente tiene detractores, y cada vez tiene más seguidores, quienes tienen lo siguiente que decir sobre el lenguaje:

#### **Eric S. Raymond, Hacker, autor de “La catedral y el bazar”**

*“Mi segunda sorpresa vino unas horas después de haber comenzado a escribir mi primer proyecto en Python, ya que descubrí que estaba generando código funcional casi tan rápido como podía teclear. [...] Una importante medida del esfuerzo de programar consiste en la frecuencia con la cual escribes algo que no se corresponde completamente con la representación mental que tienes del problema, y debes volver hacia atrás sobre tus pasos descubriendo que lo que acabas de teclear no le va a indicar al ordenador que haga lo que estas pensando. [...] Cuando escribes código funcional casi tan rápido como puedes teclear y tu tasa de error es prácticamente nula, generalmente significa que acabas de convertirte en un maestro del lenguaje. ¡Pero eso no tenía sentido, porque todavía me encontraba en el día en el que hojeas el manual buscando nuevas características del lenguaje y la librería!”*

#### **Tommy Burnette, Director técnico senior de Industrial Light & Magic**

*“Python juega un papel crucial en nuestro ciclo de producción. Sin él, un proyecto del tamaño de “Star Wars: Episodio II” hubiese sido muy difícil de asimilar. Desde renderizado de espectadores hasta la composición, pasando por tareas como el procesamiento por lotes, Python agrupa todo el proceso sin problemas.”*

#### **Philip Peterson, Ingeniero Principal, I+D en Industrial Light & Magic**

*“Python está en todas partes en ILM. Lo usamos para extender las características de nuestras aplicaciones al igual que nos proporciona un enlace entre ellas. Toda imagen generada por ordenador ha tenido a Python involucrado en alguna parte del proceso de creación.”*

#### **Peter Norvig, Director de calidad de búsqueda en Google**

*“Python ha sido una parte importante de Google ya desde nuestros comienzos, y sigue siendolo a medida que el sistema crece y evoluciona. A día de hoy docenas de ingenieros de Google usan Python, y estamos muy interesados en encontrar a gente con habilidad en este lenguaje.”*

#### **Steve Waterbury, Líder del Grupo de Software de la NASA**

*“La NASA está usando Python para implementar un repositorio CAD/CAE/PDM y modelar un sistema de gestión, integración y transformación que será el núcleo de la infraestructura para el entorno de ingeniería colaborativa de siguiente generación. Elegimos Python porque proporciona la productividad máxima, código que es limpio y fácil de mantener, librerías fuertes y extensibles (¡y cada vez hay más!), y capacidades*



*excelentes de integración con otras aplicaciones en otras plataformas. Todas estas características son esenciales para construir sistema eficientes, flexibles, escalables y bien integrados, que es exactamente lo que necesitamos.”*

Entre otros usuarios satisfechos de Python se encuentran: Yahoo, IBM, Philips, Disney, Red Hat Linux, RealNetworks, Totally Games, el Instituto Nacional de la Salud de los Estados Unidos, la división de física teórica en el Laboratorio Nacional de Los Álamos, y muchos más [27].

## 2.3 GTK y PyGTK

Para comunicarse con el usuario Tennacles necesita una interfaz gráfica sencilla de manejar. Cada vez hay más usuarios que optan por plataformas alternativas a Windows y se pasan a máquinas con MacOS o alguna variante de Unix. Este argumento impide categóricamente usar APIs de Microsoft para la implementación de la interfaz, porque sólo están disponibles en la plataforma Windows y se reduciría el número potencial de usuarios de la aplicación.

En el terreno de las interfaces multiplataforma hay dos opciones que destacan sobre las demás: Qt [32] y GTK. Qt está desarrollado por Trolltech [33], y nació siendo una librería puramente comercial. Debido a unas complicaciones con las licencias del escritorio KDE de Linux, una de las aplicaciones más famosas que usa esta librería, ofrecieron una versión GPL de Qt únicamente para Linux, que servía también en otras plataforma mientras el programa final fuese gratuito, impidiendo la generación de negocio a no ser que se compren las caras licencias de Trolltech.

Esto es suficiente razón para evitar esta librería ya que Tennacles en un futuro podría convertirse en el producto de una empresa. En cambio, GTK es una librería con licencia LGPL [19], que no impide la comercialización de los productos finales, protegiendo a su vez el patrimonio de código libre en el que está basada.

Desde el punto de vista técnico, Qt es una librería escrita en C++ con el objetivo de proporcionar a un programa una interfaz de aspecto “nativo”. Es decir, cuando se compila un programa con Qt para Windows, el aspecto visual es el de cualquier otra aplicación que use librerías como las MFC. El mismo programa compilado con Qt para Mac parecerá una aplicación de Macintosh de toda la vida, y lo mismo con Linux. No sólo cambia el aspecto, también otros aspectos de usabilidad como comportamiento de las ventanas, atajos del teclado, etc.

Qt es además una librería bastante gorda en cuanto a peso, porque añade muchas funciones de manejo de ficheros, teclado, caracteres Unicode y demás para proporcionar al programador usuario un entorno uniforme en todas las plataformas a las que quiera portar su programa. Quizás una de las características más aclamadas se la programación con “slots”, un mecanismo similar a implementar una interfaz, bajo la cual una clase que hereda de un objeto puede definir qué métodos quiere llamar antes o después que los métodos por defecto, añadiendo una ejecución encadenada de funciones. El usuario por tanto escribe una especie de meta-clase que luego es precompilada por un compilador especial de Qt y que genera el código real que engancha el comportamiento, facilitando su implementación.

GTK en cambio es una librería en C que originalmente se desarrolló para crear el programa de dibujo gráfico GIMP, y se vio que era tan útil que decidieron separarla del programa para que otros pudiesen reutilizar el código. Actualmente se dispone de la versión



 <p>Universidad de Deusto Facultad de Ingeniería</p>	<h2>2.3 GTK y PyGTK</h2>	<p>Proyecto Fin de Carrera</p>
---	--------------------------	--------------------------------

2 de GTK, la cual corrige sobre todo muchas peculiaridades de la API que eran necesarias por estar todavía GTK algo entrelazado con GIMP.

La filosofía de GTK es distinta respecto a Qt. En lugar de proveer una interfaz uniforme con el resto de aplicaciones en cada plataforma, GTK define un aspecto propio neutro que es igual en cada sistema operativo. Esto se decidió así para evitar molestias a los usuarios en cuanto a usabilidad. Mientras que con Qt el objetivo era que los usuarios que sólo conocen un sistema disfruten de la misma interfaz a la que están acostumbrados, GTK apuesta por el futuro aceptando que cada vez va a haber más plataformas diferentes en el mercado, evitando al usuario que deba aprender los detalles de cada sistema operativo si por alguna razón debe cambiarlo (algo que cada vez ocurre más en las empresas al migrar por ejemplo de Windows a Linux o Macintosh), proporcionando un único estilo de interfaz.

Por supuesto GTK no es técnicamente inferior a Qt. Aunque no se dispone de los supuestos “slots”, puesto que esto es un mecanismo reservado para la herencia de clases, GTK proporciona “señales y eventos”. Básicamente permiten emular la orientación a objetos de C++. Cuando el usuario pincha en un botón, un evento es emitido a la aplicación. El programador puede enganchar este evento conectando la señal correspondiente con la llamada a una función, que recibe como primer parámetro el elemento gráfico que ha generado la señal (claramente orientación a objetos pero desde C).

Aunque esto es un método algo más engorroso y manual de implementar una interfaz, proporciona la misma flexibilidad. Además, Tennacles al usar Python interactuará con la versión “pitónica” de GTK, que como soporta orientación a objetos es igual de sencilla de programar que la versión “pitónica” de Qt. Dado que Qt impone más restricciones en la creación de programas, GTK es la solución elegida, tanto por su libertad como por su funcionalidad.

En la sección anterior se mencionó que las funciones anónimas eran muy útiles en la construcción de interfaces. Esto es así porque para implementar una ventana, lo que se hace es heredar del objeto ventana que proporciona la librería de interfaz gráfica e implementar los métodos que queramos. En la práctica suele ser común que varios métodos de la clase sean simples y únicamente devuelvan constantes, como por ejemplo para implementar el método que cierra una ventana y solamente hace falta devolver el valor booleano verdadero para que la ventana sea efectivamente cerrada.

Las funciones anónimas pueden reemplazar las funciones habituales. Simplemente en el constructor del objeto, en lugar de asociar una señal con un método de la clase se asocia con una función anónima simple. A continuación se muestra parte del código del constructor de la ventana de bienvenida de la aplicación:



```
class Welcome(gtk.Window):

    def __init__(self):
        super(Welcome, self).__init__(gtk.WINDOW_TOPLEVEL)
        self.set_title("Lanzador Tennacles")
        self.set_border_width(10)

        # destrucción de la ventana
        self.connect("delete_event", lambda x, y, z, w: 0)
        self.connect("destroy", lambda x: tennacles.ui.close())

        # ... más código aquí ...
        self.button4 = button4 = gtk.Button("Cerrar esta ventana")
        box1.pack_start(button4, 1, 1, 0)
        button4.connect("clicked", lambda x: tennacles.ui.close())
        button4.show()
```

Aunque la clase tiene más métodos, resulta evidente que gracias a las funciones anónimas no hace falta implementar el método que se encarga de controlar el evento de borrado y destrucción de la ventana, al igual que el que controla la pulsación de uno de los botones. Así, la clase final queda más limpia porque tiene menos métodos superfluos, y los que están, son los que realizan acciones más complejas e importantes. En otras palabras, las funciones anónimas ayudan a reducir el ratio código superfluo/código útil.

## 2.4 Allegro y otras librerías complementarias

Dadas las especificaciones de Tennacles, es necesario cargar datos gráficos y sonoros en diversos formatos. Para hacer esta tarea se recurrirá a Allegro, una librería para programadores de C/C++ orientada al desarrollo de videojuegos, distribuida libremente, y que funciona en las siguientes plataformas: DOS, Unix (Linux, FreeBSD, Irix, Solaris), Windows, QNX y BeOS (la versión MacOS está en estado alpha). Tiene muchas funciones de gráficos, sonidos, entrada del usuario (teclado, ratón y joystick) y temporizadores. También tiene funciones matemáticas en punto fijo y coma flotante, funciones 3d, funciones para manejar ficheros, ficheros de datos comprimidos y una interfaz gráfica.

No obstante, Tennacles sólo usará por el momento las rutinas encargadas de cargar y salvar ficheros gráficos y sonoros. De hecho, para poder soportar formatos como JPEG u OGG, se recurrirá a otras librerías que extienden la funcionalidad original de Allegro:

### JPGalleg [21]

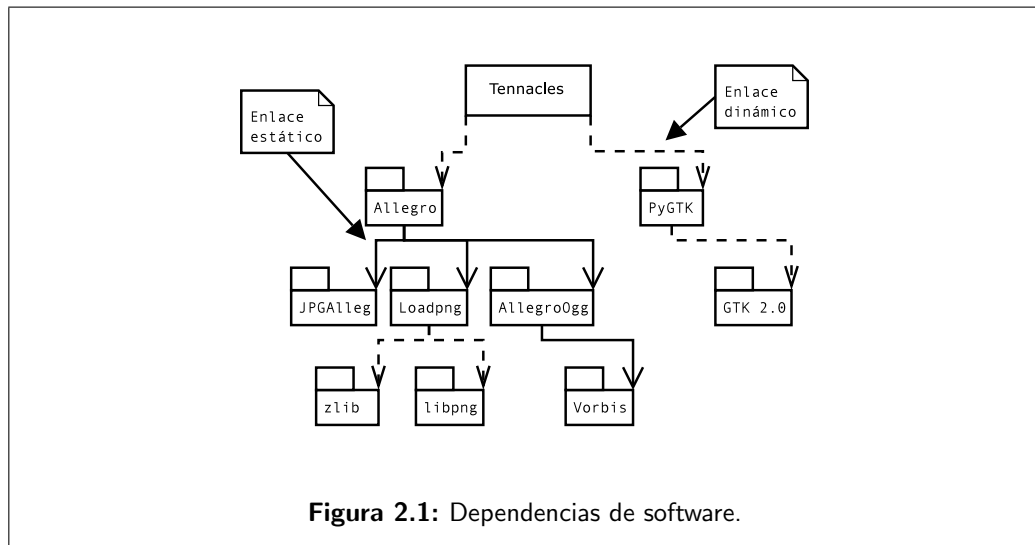
Esta librería extiende las capacidades de carga de ficheros gráficos, haciendo del formato gráfico JPEG un tipo nativo de Allegro. La librería implementa de forma propia un cargador de ficheros JPEG, por lo que no tiene dependencias externas.

### Loadpng [35]

Esta librería extiende las capacidades de carga de ficheros gráficos, haciendo del formato gráfico PNG un tipo nativo de Allegro. La librería hace de interfaz contra la librería de software libre 'libpng', por lo que ésta también será enlazada con la aplicación, la cual a su vez enlaza con 'zlib' para las rutinas de compresión.

### AllegroOgg [10]

Esta librería extiende las capacidades de carga de ficheros sonoros, haciendo del




formato de audio OGG un tipo nativo de Allegro. La librería hace de interfaz contra la librería de software libre ‘libogg’ proporcionada por la empresa Vorbis [34], por lo que ésta también será enlazada con la aplicación.

Las licencias de Allegro y estas otras librerías satélite son o bien LGPL o de dominio público, por lo que no plantean ningún conflicto de licencias o distribución del programa final, y todas son multiplataforma.

El grafo final de dependencias de software queda como se ve en la figura 2.1. Sólo se muestran los paquetes con los que habrá que tratar manualmente. Por ejemplo, GTK depende de otras librerías como Pango [25] o Glib [9], pero no se especifican porque sólo se puede acceder a ellas a través de GTK, nunca directamente.



 <p>Universidad de Deusto Facultad de Ingeniería</p>	<p>3 Diseño de las ventanas de la interfaz</p>	<p>Proyecto Fin de Carrera</p>
---	--	--------------------------------

## CAPÍTULO 3

# Diseño de las ventanas de la interfaz

### 3.1 Bienvenida

La ventana de bienvenida tiene el propósito de saludar al usuario en cuanto arranque la aplicación, mostrando las operaciones más frecuentes:

#### **Crear un nuevo repositorio**

Esta opción llevará a la ventana del sistema de archivos que muestra el contenido del soporte físico para que el usuario seleccione el lugar donde quiere establecer el fichero central que controla su proyecto. Después se le pedirán unos datos generales para el proyecto y finalmente se mostrará el gestor de recursos con el proyecto iniciado.

#### **Abrir un repositorio recientemente usado**


Debajo de esta opción habrá una lista con un máximo de cuatro entradas, siendo cada entrada el título de los repositorios recientemente usados. Al seleccionar un elemento de la lista y pinchar en el botón, o hacer doble click en el elemento de la lista se abrirá el proyecto indicado. Habrá un último (quinto) elemento en la lista con el texto “Un repositorio no mostrado en esta lista”. Esta opción abrirá la ventana del sistema de archivos para que el usuario busque el proyecto que desee y lo abra.

#### **Abrir manual**

El manual de Tennacles estará disponible en formato HTML, y esta opción localizará y lanzará la página frontal con el navegador registrado en el sistema. Si el SO no soporta lanzar una aplicación asociada a contenido HTML, Tennacles ofrecerá al usuario la posibilidad de seleccionar un binario específico con el que abrir el manual, que podrá ser un navegador o cualquier otro programa capaz de procesar y representar HTML.

#### **Cerrar la ventana de bienvenida**

Esta es una opción análoga a cerrar la ventana pinchando en el icono de cierre usado

 <p>Universidad de Deusto Facultad de Ingeniería</p>	<p>3 Diseño de las ventanas de la interfaz</p>	<p>Proyecto Fin de Carrera</p>
---	--	--------------------------------

por el SO que decora la ventana. Cierra la ventana de bienvenida sin afectar otras ventanas ya abiertas a través de esta interfaz.

## 3.2 Ventana de gestión de recursos

Esta es la ventana central de cada proyecto. Todas las acciones de la herramienta comenzarán desde aquí. La ventana consiste de una barra de título que muestra el título del repositorio y los botones habituales que usa el SO para decorar la ventana, una barra de menús con algunas opciones, un área de botones y dos áreas de visualización, una usada para mostrar un árbol y otra para mostrar los recursos como iconos o listas.

Dado que esta es la ventana principal, su cierre supone la terminación del programa, obligando al usuario a que mantenga esta ventana siempre abierta. Dependerá del gestor de ventanas usado el que esta ventana pueda ser minimizada o puesta en segundo plano.

El área izquierda de la ventana muestra una estructura jerárquica con forma de árbol que replica la estructura de ficheros del sistema donde se almacena el repositorio. Si el usuario intenta reescalar el gestor de recursos, el área de la izquierda mantendrá su anchura, mientras que el área derecha se podrá adaptar al nuevo tamaño escogido por el usuario.

Cuando un nivel jerárquico es seleccionado en el área izquierda, el área derecha muestra los contenidos de esa rama en forma de lista detallada. Dado que el listado muestra tipos mixtos de datos, sólo las dos primeras columnas (identificador y descripción) son comunes, la tercera columna es de contenido libre: un recurso de imagen añadirá datos sobre el tamaño de la misma, mientras que un recurso de sonido indicará si es estéreo o no entre otras características.

Hacer doble click en un recurso accionará la iniciación de un programa asociado con su edición, y hacer click con el botón derecho mostrará un menú emergente con un conjunto variado de opciones como abrir, borrar, ver propiedades, etc.

A continuación se describen las opciones de los menús disponibles en la parte superior de la ventana.

### 3.2.1 Menú “Repositorio”

#### Actualizar recursos


Esta opción invocará el refresco de los recursos físicos del disco duro y actualizará la versión que se tiene de ellos en memoria. Para hacer esto de una forma óptima el programa sólo actualiza aquellos ficheros en los que haya cambiado su fecha de modificación.

#### Seleccionar plugin

Aparecerá un menú para permitir al usuario seleccionar un plugin de exportación. Si no hay plugins disponibles, una opción del diálogo permitirá al usuario añadir plugins no encontrados en los directorios por defecto que busca Tennacles.

#### Configurar plugin: xxx

La cadena “xxx” será reemplazada por el nombre del plugin de exportación seleccionado actualmente. Seleccionar esta opción llamará una función del plugin que permitirá al usuario personalizar la exportación. Este plugin es considerado por tanto como un programa que se ejecuta de forma independiente a Tennacles aunque usando la API de gestión de recursos para salvar opciones en el repositorio. Por esta

 <p>Universidad de Deusto Facultad de Ingeniería</p>	<h3>3.3 Atributos del proyecto</h3>	<p>Proyecto Fin de Carrera</p>
---	-------------------------------------	--------------------------------

razón, podrá usar la interfaz de diálogo que considere apropiada, que no tiene que ser consistente con la de Tennacles.

#### **Exportar con: xxx**

Ejecuta el plugin con las opciones seleccionadas previamente o los valores por defecto. De nuevo, esto significa llamar a un programa totalmente externo sobre el cual Tennacles no tiene control alguno. Esta llamada no es bloqueante: mientras el plugin de exportación está funcionando el usuario sigue pudiendo trabajar con Tennacles (lo cual podría ser pernicioso para el plugin si los datos se modifican mientras éste se ejecuta).

#### **Propiedades**

Esta opción abre la ventana de atributos del proyecto, lo que permite al usuario modificar su nombre, el título del proyecto o su descripción.

#### **Cerrar programa**

Cierra todas las ventanas del programa y finaliza la aplicación.

### 3.2.2 Menú “Nuevo recurso”

#### **Nueva animación**

Genera un nuevo recurso lógico abstracto que representará una animación cualquiera. Envolverá a otros recursos, principalmente físicos.

#### **Nueva acción lógica**

Genera un nuevo recurso lógico que contendrá la secuencia de pasos que debe realizar la acción lógica, que puede ser asociada a un evento de la aventura.

#### **Nuevo actor**

Genera el recurso lógico de actor, necesario para representar objetos y personajes con los que se pueda interactuar.

#### **Nueva escena**

Genera una escena, que es una agrupación de todos los recursos lógicos y físicos anteriores y que es la unidad de trabajo básica en la aventura (sin escena no se ve nada).

### 3.2.3 Menú “Ayuda”

#### **Manual**


Invoca al navegador web que abrirá el manual de usuario en formato HTML.

#### **Acerca de Tennacles**

Muestra un diálogo con información sobre el propio programa e información de contacto con el autor.

## 3.3 Atributos del proyecto

Después de seleccionar la situación física de nuestro proyecto en el disco duro, es necesario introducir algunos parámetros globales del proyecto, como su título, descripción, autores y resolución física, lo cual se hará a través de cajas de texto. El botón cancelar

 <p>Universidad de Deusto Facultad de Ingeniería</p>	<p>3 Diseño de las ventanas de la interfaz</p>	<p>Proyecto Fin de Carrera</p>
---	--	--------------------------------

evitará la creación del proyecto y se volverá a la ventana de bienvenida si esta ventana es invocada durante la creación. En caso de haber sido abierta más tarde, únicamente evitará que los cambios realizados en ella se hagan efectivos.

Las aventuras gráficas dependen de una resolución física constante, y sólo se puede modificar en la creación del proyecto, más tarde los campos de la ventana que muestran la resolución no serán editables.

### 3.4 Selección de plugin

Los plugins son externos a Tennacles, pueden ser escritos por cualquiera y solamente se requiere que sigan la interfaz especificada en el capítulo 9. Tennacles buscará los plugins en unos directorios por defecto y los cargará, mostrando una lista de ellos al usuario.

El diálogo contiene dos áreas: el área izquierda muestra una lista de los plugins disponibles que se han cargado hasta el momento. El área de la derecha se subdivide en dos secciones verticales: la parte superior muestra información sobre el plugin actualmente seleccionado, o un mensaje que indica que no hay plugin seleccionado o que su información no puede ser mostrada. La sección inferior del área derecha contendrá unos cuantos botones con las siguientes opciones:

#### Refrescar directorios

Cada vez que el usuario abre esta ventana las rutas que contienen los plugins son analizadas. Pero si la siguiente opción se usa para alterar las rutas o el usuario modifica el código del plugin de forma externa, la activación de este botón forzará un refresco de las rutas de los plugins y del código disponible, útil cuando un desarrollador está depurando la interfaz de su plugin o el usuario acaba de copiar un nuevo plugin a uno de los directorios buscados.

#### Modificar directorios de búsqueda


Esta opción abre un nuevo diálogo con una lista de las rutas que Tennacles analiza en la búsqueda de plugins. Debajo de la lista estarán los siguientes botones: “Añadir directorio”, “Eliminar directorio”, “Salvar cambios” y “Cancelar cambios”. El primer botón abrirá la ventana del sistema de archivos, pero en lugar de seleccionar un directorio que contiene un repositorio, el usuario puede añadir cualquier directorio. La segunda opción sólo estará activa cuando el usuario seleccione un directorio de la lista de directorios buscados, entrada que será eliminada si se acciona el botón. Las dos últimas opciones permiten salvar los cambios realizados en la lista de rutas buscadas o cancelar todas las operaciones, pudiendo volver al estado anterior a la apertura de esta ventana.

### 3.5 Editor de escenas

El editor de escenas es el lugar donde ocurre la mayor parte del tiempo de desarrollo de una aventura gráfica. Aquí, el usuario puede diseñar, componer la escena seleccionada de la aventura. Si se desean crear una nueva escena, hay que hacerlo desde los menús de la ventana de gestión de recursos.

El editor gráfico de escenas consiste en una pequeña barra de botones en la parte superior, y la representación gráfica de la escena en el área inferior. Si la escena no cabe, habrá barras de desplazamiento vertical y horizontal. Las escenas se componen de varias



 <p>Universidad de Deusto Facultad de Ingeniería</p>	<h3>3.5 Editor de escenas</h3>	<p>Proyecto Fin de Carrera</p>
---	--------------------------------	--------------------------------

capas individuales, cada una de las cuales contiene una información muy específica. Las operaciones del usuario nunca se aplicarán a más de una simultáneamente, para seleccionarla se usará el listado disponible en la barra de tareas que contendrá las siguientes opciones:

- Capa de fondo, usada para definir los gráficos on interactivos de la escena. Por defecto visible.
- La capa Z-buffer de fondo, que especifica qué pixels del fondo deberían aparecer en frente de los demás objetos. Por defecto oculta.
- La capa de desplazamiento, indica las zonas por las que pueden desplazarse los personajes. Por defecto oculta.
- La capa de zonas calientes, donde se definen por ejemplo los enlaces con otras escenas. Por defecto oculta.
- La capa de actores (personajes y objetos). Por defecto visible.


Las capas serán representadas gráficamente en ese orden de atrás hacia adelante. Un botón de radio colocado al lado de la lista de selección permitirá visualizar u ocultar la capa seleccionada. Esto es necesario para capas con información de Z-buffer, cuyos datos representados de forma opaca ocultarían la capa de fondo, algo que no siempre es deseable. La capa de Z-buffer, de desplazamiento y zonas calientes sólo proporcionan información, y no serán visibles en la aventura gráfica generada.

Cada capa tiene definida un conjunto de operaciones que se pueden realizar sobre ella. La capa de Z-buffer en realidad no es más que un gráfico asignado desde el gestor de recursos. Los datos de esta capa no pueden ser editados, lo que significa que el Z-buffer deberá ser un recurso previamente creado por el usuario. Si no hay Z-buffer asociado con la escena cuando el juego sea exportado, será tarea del plugin controlar correctamente esta omisión (la cual podría ser tratada de forma aceptable creando una capa por defecto y avisando al usuario de esto). Las capas de Z-buffer, desplazamiento y zonas calientes podrán visualizarse con una opacidad completa o del 50 %, a parte de ser ocultadas completamente.

Las otras dos capas (capa de fondo y capa de personajes y objetos) permiten la manipulación o composición de los recursos disponibles. Por ejemplo, un fondo puede estar compuesto de tres recursos estáticos y dos recursos animados, todos pertenecientes a la misma capa. De forma equivalente, la capa de personajes y objetos puede contener un hombre caminando por la escena, un actor principal con el que ego puede hablar y el propio ego. Las operaciones básicas que el usuario puede realizar son la colocación de recursos y la modificación de sus atributos.

La colocación de los recursos es sencilla: el usuario arrastra un recurso desde la ventana de gestión de recursos a la ventana del editor de escena, y un rectángulo representando el objeto aparecerá en el editor de escenas. Cuando el usuario suelte el botón del ratón, finalizando el movimiento de arrastrado, el recurso será renderizado y el usuario podrá continuar moviéndolo por la pantalla para corregir su colocación.

Pinchando con el botón derecho en el recurso hará aparecer un menú desplegable con cierta información básica sobre el recurso y un botón “Atributos”, el cual abrirá una ventana con los atributos del recurso seleccionado. Hacer doble click en el recurso abrirá el módulo que está asociado con la modificación del recurso, como puede ser el módulo de conversaciones para un personaje.

 <p>Universidad de Deusto Facultad de Ingeniería</p>	<p>3 Diseño de las ventanas de la interfaz</p>	<p>Proyecto Fin de Carrera</p>
---	--	--------------------------------

### 3.6 Ventana de atributos

La ventana de atributos no está siempre visible, aparece cuando el usuario lo desea pinchando con el botón derecho en un recurso, ya sea en el gestor de recursos, en el editor de escenas, o cualquier otra parte y selecciona la opción “Ver propiedades”.

Una vez abierta esta ventana, permanecerá abierta hasta que el usuario cierre la ventana o su ventana padre del editor de escenas. Sólo puede haber una ventana de atributos, lo que significa que si el usuario selecciona otro elemento de la escena, esta ventana pasará a mostrar directamente los atributos de la nueva selección. Si la selección se pierde o el objeto seleccionado no tiene atributos de ningún tipo, la ventana permanecerá abierta y reflejará este estado con el mensaje “No hay atributos disponibles”.

La ventana es independiente de otras. Aunque sea abierta desde el editor de escenas, si éste se cierra la ventana de atributos seguirá abierta aunque sin contenido.

Los atributos se representarán como un listado de las características del elemento. Dependiendo del tipo de elemento se verán sus propiedades físicas o lógicas. La descripción detallada de los atributos mostrados para cada elemento se encuentra en el capítulo 4.

### 3.7 Editor de acciones lógicas

Las acciones lógicas no tienen una representación gráfica en las escenas, precisamente porque sólo trabajan a nivel lógico. Las acciones están a la espera durante todo el juego y reaccionan ante eventos generados por la aventura, momento en el que se activan y determinan si deben modificar algo.

Para expresar una acción lógica se usarán simples diagramas de flujo, principalmente con dos elementos: condiciones (representadas por rombos) y acciones (representadas por rectángulos). El comienzo del diagrama de flujo será representado por un círculo del cual saldrá la flecha inicial del diagrama, donde se indicará el tipo de evento que activa la evaluación de la acción lógica.

El editor de acciones lógicas sólo permite modificar una acción, la cual debe haber sido creada previamente desde el gestor de recursos. No obstante es posible abrir varios editores de acciones lógicas simultáneamente para modificar varias acciones.


Dado que la representación del diagrama de flujo puede ocupar fácilmente más espacio que el visible en la ventana, aparte de los deslizadores habituales, la ventana centrará cualquier condición o acción que sea seleccionada por el usuario, facilitando la navegación.

La ventana del editor de acciones es simple, se divide únicamente en dos secciones, el área superior muestra el diagrama de flujo, y la parte inferior permite modificar el texto contenido en el elemento seleccionado. No hay opciones de salvar o guardar, se trabaja directamente sobre el recurso lógico y las modificaciones son inmediatas, sin necesidad de cerrar la ventana para que tengan efecto.

#### 3.7.1 Condiciones

Una condición contendrá una expresión cuya evaluación devolverá un valor de verdadero o falso. La expresión puede hacer comparaciones entre enteros, flotantes, y cadenas (usadas para identificar recursos lógicos complejos), que serán variables globales del juego.

Se pueden unir varias expresiones simples para crear una expresión compleja mediante las palabras clave AND y OR. El resultado de la evaluación de la expresión se puede negar

 <p>Universidad de Deusto Facultad de Ingeniería</p>	<h3>3.8 Editor de actores</h3>	<p>Proyecto Fin de Carrera</p>
---	--------------------------------	--------------------------------

mediante la palabra clave NOT. Las expresiones se pueden agrupar mediante paréntesis. Una expresión se puede escribir en el área de edición en varias líneas, siendo todas concatenadas antes de evaluar la expresión.

#### 3.7.2 Acciones

Las acciones pueden ser operaciones matemáticas sobre las variables globales del juego (suma, resta, asignación, etc). También pueden ser invocaciones a otras acciones, en cuyo caso basta con escribir su nombre. En este caso, el motor ejecutará la acción indicada y cuando finalice continuará con la acción en curso. Las acciones no devuelven ningún valor por sí mismas, así que no pueden ser usadas dentro de otras expresiones matemáticas que asignan valores. Cada acción debe escribirse en una única línea para que pueda ser identificada correctamente.

### 3.8 Editor de actores

---

El editor de actores muestra los datos del actor seleccionado para que el usuario pueda modificarlos. La barra de título de la ventana informará del actor que está siendo modificado. La ventana se divide en dos áreas, la izquierda permite seleccionar una categoría de los atributos del actor. Seleccionando una categoría de la izquierda aparecerá en el área derecha los tipos de atributos que se pueden modificar.

Sólo se pueden modificar los atributos que no se pueden cambiar desde otro editor. Es decir, el editor de actores únicamente permite asociar las animaciones y acciones lógicas del desplazamiento, habla, etc, pero no permite modificar la posición en pantalla, ya que la posición depende del editor de escenas.

La asignación de atributos se hace al igual que en otras ventanas arrastrando recursos desde el gestor de recursos hasta el atributo. Para borrar un recurso se puede usar la tecla 'suprimir'.

### 3.9 Editor de diálogos

---

Esta ventana sólo se abre cuando hay que modificar el diálogo que está asociado a un personaje. La ventana se divide horizontalmente en tres áreas. La superior se corresponde con la vista jerárquica de los nodos del diálogo, siendo el nodo raíz la primera frase que nos dice el personaje al acercarnos a él. Debajo, aparecerá la frase en una caja de texto para que la podamos modificar junto al identificador preasignado del nodo creado. En la parte inferior, aparecerá una barra de botones que nos permitirá definir los atributos del nodo:

#### Acabar diálogo


Este nodo es el último de la conversación.

#### Crear nodo hijo


Este botón permite añadir respuestas al nodo actualmente seleccionado, pero sólo si se trata de respuestas adicionales que debe responder Ego.

#### Asociar acción lógica

Cuando la conversación pasa por este nodo se ejecuta incondicionalmente la acción

 <p>Universidad de Deusto • • • • Facultad de Ingeniería</p>	3 Diseño de las ventanas de la interfaz	Proyecto Fin de Carrera
---	---	----------------------------

lógica. Por ejemplo, ésta podría modificar una variable lógica para desbloquear un acertijo en otro lugar de la aventura.

 <p>Universidad de Deusto Facultad de Ingeniería</p>	<p>4 Tipos de recursos</p>	<p>Proyecto Fin de Carrera</p>
---	----------------------------	--------------------------------

## CAPÍTULO 4

# Tipos de recursos

Este capítulo de la documentación explica los tipos de recursos que podrán ser manejados con Tennacles, tanto los recursos físicos (ver figura 4.1) como los recursos virtuales simples (ver figura 4.2) y complejos (ver figura 4.3) creados por la herramienta. Por cada recurso se indicará la forma en la que Tennacles trata este recurso, sus atributos y las operaciones especiales que deban definirse sobre el mismo. Ambos tipos de recursos derivan de una clase llamada Recurso, la cual únicamente tendrá los dos siguientes atributos:

- **Identificador o clave única.** Todos los recursos físicos deben estar dentro de la jerarquía de directorios que comienza en el repositorio, así que no tiene sentido almacenar su ruta completa. Su ruta relativa además sirve para identificar al fichero de forma única y absoluta, dado que no puede haber dos ficheros con la misma ruta relativa.

Los recursos lógicos no tienen representación física en la jerarquía de directorios ya que se almacenan en el fichero de repositorio y son creados por el usuario. Por esta razón, su identificador será una cadena de caracteres alfanuméricos elegidos al azar, y para diferenciar su identificador de los recursos físicos, el primer carácter será la barra “/”, carácter que no puede aparecer de forma natural en el nombre relativo de un fichero, pues es el carácter usado para separar directorios en las rutas bajo sistemas Unix.

El identificador de los recursos lógicos puede tener la forma de una ruta relativa de fichero “virtual”, ya que así puede ayudar al usuario a agrupar recursos jerárquicamente para que sean manejables.

- **Nombre simbólico o descripción.** Aunque los recursos lógicos pueden tener como identificador cualquier cadena que desee el usuario, el identificador de un recurso físico puede ser ineficiente de cara al usuario para indicar qué es (ejemplo: “gfx/img0032.bmp”). El nombre simbólico describe al recurso,

y puede ser cualquier cadena de texto introducida por el usuario, como por ejemplo “Paso tercero de la secuencia de movimiento noroeste del protagonista”. Por defecto

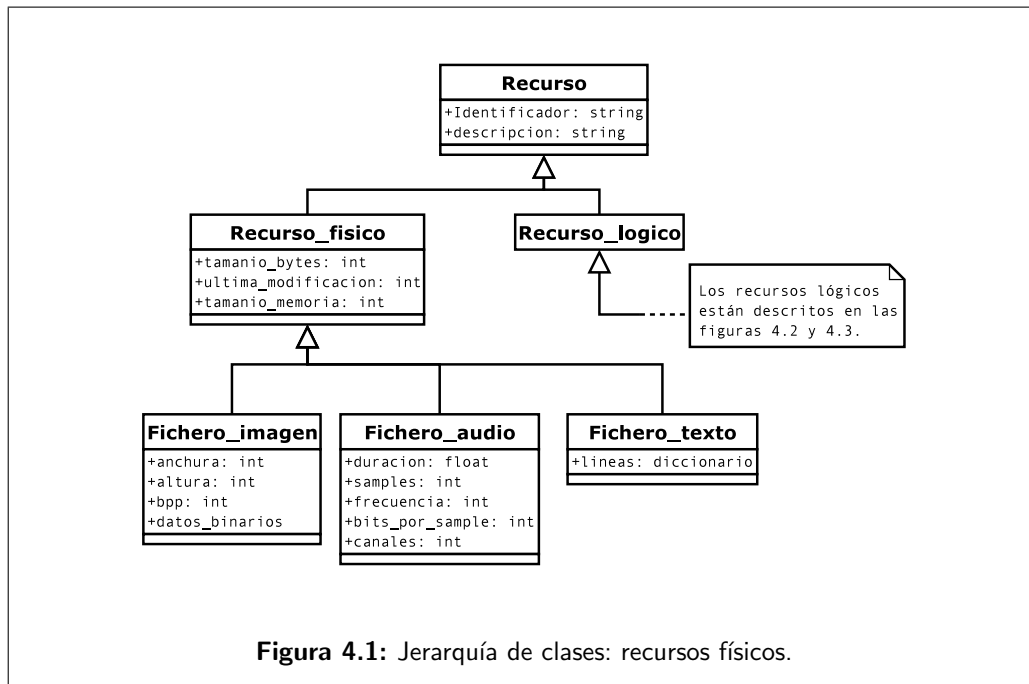


Figura 4.1: Jerarquía de clases: recursos físicos.

será la cadena vacía.


## 4.1 Recursos físicos

Se trata de los recursos que el usuario puede manejar de forma externa al editor ya que existirán como ficheros individuales. Debido a esta característica, todos los recursos físicos compartirán ciertos atributos, que serán:

- Tamaño físico del fichero origen del recurso medido en bytes.
- Última fecha de modificación del archivo, necesaria para saber si hay que actualizar el recurso actualmente en memoria a partir del fichero debido a un cambio externo producido por el usuario.
- Tamaño en memoria del recurso: cantidad de bytes que ocupará el recurso una vez cargado en memoria. Esta cantidad no es siempre obvia a primera vista, dado que algunos ficheros usan compresión y otros no. Además, esta cifra será más aproximada a la memoria consumida por Tennacles, ya que un motor podría tratar el recurso en memoria de forma diferente (por ejemplo, manteniéndolo comprimido en memoria y descomprimiéndolo cuando sea necesario bajo demanda).

### 4.1.1 Ficheros de imágenes

Los gráficos son el principal recurso usado por las aventuras gráficas. El usuario podrá cargar gráficos contenidos en cualquiera de los siguientes formatos: BMP, PCX, TGA, LBM, JPEG y PNG, básicamente imágenes basadas en pixels, no vectoriales. Los gráficos pueden tener un tamaño únicamente limitado por la memoria del ordenador, y una profundidad

 <p>Universidad de Deusto Facultad de Ingeniería</p>	<h2>4.1 Recursos físicos</h2>	<p>Proyecto Fin de Carrera</p>
---	-------------------------------	--------------------------------

gráfica de 8, 15, 16 o 24 bits. Tennacles convertirá internamente todas las profundidades a 24 para no tener conflictos de paleta durante la representación de los gráficos.

En el caso de imágenes de 32 bits (24 bits por color, 8 bits para canal alpha o de translucidez), el canal alpha será tenido en cuenta a la hora de representar el gráfico. En las imágenes de 8 bits, se permitirá al usuario especificar si uno o más colores de la imagen son totalmente transparentes, lo que equivale a un canal alpha de 1 bit.

Por lo tanto los atributos físicos de cada imagen serán:

- Tamaño del gráfico, anchura y altura medida en pixels.
- Profundidad de color, medida en bits por pixel. Posibles valores: 8, 15, 16, 24, 24+8 (canal alpha de 8 bits) y 255 - x (imagen de 8 bits de la cual x colores se han seleccionado como transparentes).
- Datos binarios. Éstos no son obtenidos del recurso hasta que no son necesitados por la rutina de dibujado de la clase Escena, y son la representación binaria del gráfico en una cadena de enteros en formato RGBP.

### 4.1.2 Ficheros de audio

Los ficheros de audio podrán estar contenidos en cualquiera de los siguientes formatos: WAV, OGG y MIDI. Los detalles específicos del sonido (como los Hz, la cantidad de bits usados para muestrear el sonido, la duración, etc) serán mostrados. Tennacles no impone ninguna restricción de estas características puesto que no los reproducirá directamente, únicamente los almacenará y relacionará con otros elementos del juego, sin cargarlos en memoria por completo. A la hora de exportar el audio será el plugin el encargado de convertir o discriminar los sonidos que no puedan ser usados debido a sus características técnicas.


Las características mostradas por Tennacles serán:

- Duración del sonido en segundos.
- Frecuencia en Hz del sonido.
- Número de bits con el que fue muestreado el sonido, 8 o 16.
- Número de canales que tiene el sonido indicado como un número, que será normalmente 1 o dos para sonidos monofónicos o estereofónicos.

### 4.1.3 Ficheros de texto

Aunque Tennacles permitirá modificar los diálogos de los personajes y cualquier otro tipo de texto desde la propia herramienta, es posible usar un programa externo que genere ficheros de texto que luego Tennacles pueda procesar. Estos ficheros de texto pueden ser asociados luego a los diálogos de los actores de la aventura mediante identificadores numéricos. Una ventaja directa de este enfoque es la posibilidad de traducir los textos de una aventura gráfica entera alterando únicamente estos recursos: tras una recompilación se podrá disfrutar del nuevo idioma en el juego.

Tennacles puede importar dos tipos de ficheros de texto, de los cuales extraerá “líneas de diálogo” individuales:

 <p>Universidad de Deusto Facultad de Ingeniería</p>	<p>4 Tipos de recursos</p>	<p>Proyecto Fin de Carrera</p>
---	----------------------------	--------------------------------

- Texto plano. Estos ficheros deben estar codificados en UTF-8 y tener una estructura interna concreta: toda línea de diálogo debe comenzar con un número entero entre corchetes al comienzo de una línea de texto del fichero. Todo lo que venga después será considerado parte de esa línea de diálogo, hasta que se encuentre otra línea del fichero que comience con otro número entre corchetes. Los números entre corchetes son usados para identificar de forma unívoca las líneas de diálogo. Ejemplo de contenido en uno de estos ficheros:

```
[1]Bueno, yo no estoy de acuerdo
[2]El ojo se le puso rojo de tal modo,
que tuvieron que llevarlo a urgencias
[3]Cuánto dolor...
Cuánto sufrimiento...
```

- Texto XML. Los ficheros XML definen su propia codificación al comienzo, y si no lo hacen, se asumirá que usan la codificación Latin1 ISO-8859-1. El fichero XML debe tener una raíz única de la cual surgen todos los elementos individuales, cada uno de los cuales tendrá un número identificador como atributo de nodo. Ejemplo contenido en uno de estos ficheros:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<dialog_lines>
  <line number="1">Bueno yo</line>
  <line number="2">Tranquilo
majete</line>
  <line number="3">Cuando no podría</line>
</dialog_lines>
```

Tennacles considerará este tipo de recursos como “almacenes” de elementos individuales (líneas de diálogo), y por tanto más tarde se hará referencia a la línea de diálogo contenida en uno de estos recursos. Es por esto que hace falta identificar las líneas de forma inequívoca:

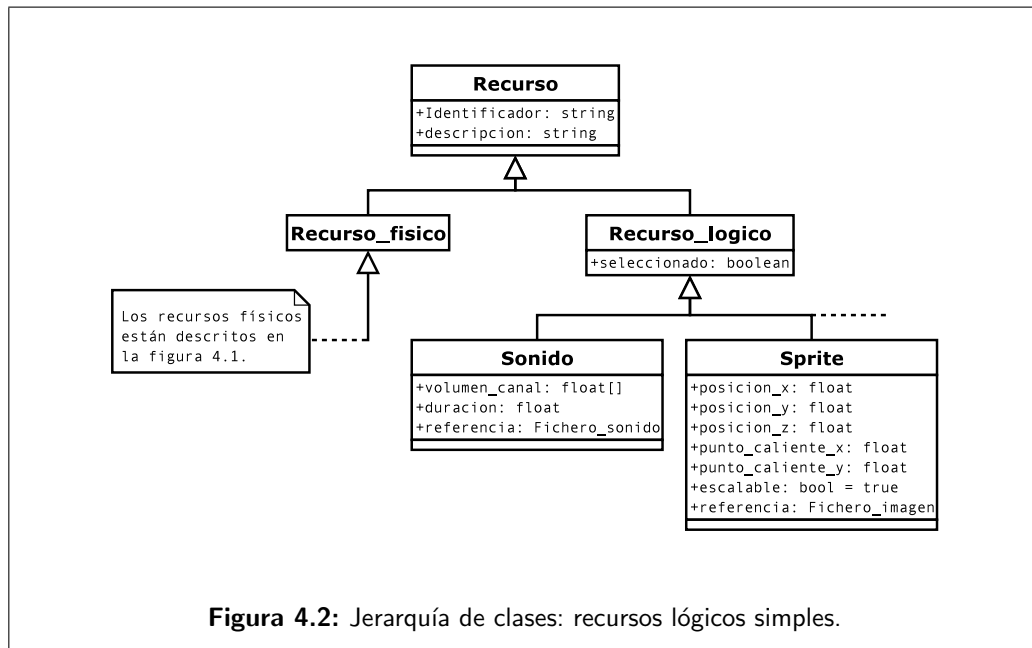
- El fichero de texto contiene muchas líneas y sólo hace falta una de ellas.
- Permite al usuario mantener cualquier orden interno en el fichero: se pueden añadir líneas al principio, al final, en medio, y todo esto no afectará a Tennacles. Gracias a los identificadores se seguirá pudiendo hacer referencia a la línea de texto apropiada sin importar su localización física en el fichero.

Por lo tanto, un recurso de fichero será representado en memoria como un diccionario que contendrá todas las líneas individuales de diálogo accesibles usando el identificador como clave. Hay una explicación más detallada en el capítulo 6.

## 4.2 Recursos lógicos

Los recursos lógicos controlados por Tennacles son todos aquellos que por un lado no tienen representación física en el disco duro, pues se almacenan en el fichero central del





repositorio, y porque suelen ser meros contenedores que hacen referencia a recursos físicos añadiendo algunos atributos útiles de cara a la aplicación y/o aventura gráfica. Dentro de los recursos lógicos hay dos tipos: los recursos lógicos simples que suele envolver a los recursos físicos creando “instancias” de éstos añadiéndoles más información (ver figura 4.2) y los recursos lógicos complejos puramente abstractos que actúan como contenedores de otros recursos lógicos y físicos (ver figura 4.3). Todos los recursos son susceptibles de ser seleccionados por el usuario en la interfaz gráfica, y ésta es la única variable que comparten todos ellos por herencia. Primero se describen los recursos lógicos simples.


### 4.2.1 Sprites

Los sprites son los elementos gráficos con información adicional básica necesaria para poder ser representados en pantalla. Añaden por tanto los siguientes atributos al objeto:

- Posición X, Y, Z. Cada valor representa una coordenada en el espacio. La coordenada X es horizontal, y los valores se incrementan a medida que se progresa hacia la derecha. La coordenada Y es vertical, y los valores se incrementan a medida que se progresa hacia abajo. La coordenada Z no existe como tal en la pantalla física, pero se usa a nivel lógico para expresar la profundidad del sprite. En este eje virtual, los valores se incrementan a medida que se progresa desde el fondo de la pantalla hacia el usuario.

Los valores pueden ser números reales (negativos, cero o positivos) que toman cualquier valor posible dentro del rango limitado por el hardware. No obstante, la zona visualizada de la aventura tiene a nivel físico unos márgenes que comienzan en el (0, 0) para la esquina superior izquierda y otro valor positivo en ambos ejes (seleccionado de antemano por el usuario al crear el repositorio) como esquina inferior derecha. Todo sprite que no tenga al menos una esquina dentro del rango especificado estará oculto para el jugador. Los valores del eje Z sólo se usan para determinar



 <p>Universidad de Deusto Facultad de Ingeniería</p>	<h2>4.2 Recursos lógicos</h2>	<p>Proyecto Fin de Carrera</p>
---	-------------------------------	--------------------------------

el orden de dibujado de los sprites cuando dos o más ocupan la misma área bidimensional. En este aspecto, si un sprite tapa completamente a otro y tiene una  $Z$  mayor, el sprite de  $Z$  menor no llegaría a ser dibujado.

- **Punto caliente.** El punto caliente es el pixel gráfico que se asocia con la posición lógica, siendo por lo tanto un valor bidimensional ( $X$  e  $Y$ ). Sabiendo que las coordenadas de los sprites siguen el sistema de coordenadas de la pantalla, un punto caliente de valor  $(0, 0)$  indicaría que el pixel superior izquierdo del sprite debe ser dibujado en la posición lógica. Un sprite de dimensiones  $(100, 100)$  cuyo punto caliente sea  $(50, 50)$  se representará centrado gráficamente sobre su posición lógica correspondiente.

Tennacles por defecto asumirá que el punto caliente de todos los sprites es el pixel central de la parte inferior del gráfico, valor que podrá ser modificado por el usuario.

- **Escalable.** Por defecto los sprites son escalables. A medida que se mueven por el escenario su tamaño cambia para dar una sensación de profundidad. Puede haber sprites que por su tamaño o características (elementos voladores como una paloma) perderían calidad al ser escalados o el escalado no les aportaría más realismo. En estos casos se puede desactivar su escalado gráfico.

### 4.2.2 Sonidos


Un sonido reproducible es un recurso sonoro con los siguientes atributos adicionales:

- **Volumen por canal.** Un valor de 0 (silencio) hasta 100 (máximo volumen) indica el volumen relativo que debe tener un sonido al ser reproducido. Así, si el volumen de la tarjeta de sonido está a un 50%, el sonido que tenga un valor de 100 sonará en realidad a un 50%, y un sonido con el valor de 50 sonaría al 25%. Este atributo está duplicado: un valor sirve para el canal izquierdo, y otro para el derecho. En el caso de sonidos monofónicos, permite reproducir el sonido por ambos canales o uno de ellos (ajustando el otro a 0). En el caso de sonidos estereofónicos, indica el volumen de cada canal.
- **Duración.** Aunque un sonido tiene una duración máxima indicada por la longitud del tamaño del fichero, con este valor se puede hacer que sea reproducido sólo un fragmento inicial. El valor se indica en segundos, y si no lo modifica el usuario, por defecto siempre será igual a la duración del fichero físico.

### 4.2.3 Acciones lógicas

Las acciones lógicas son disparadores activados por uno o más eventos generados por las acciones del jugador. En cuanto se activan, el motor evalúa las acciones que están “a la escucha” del evento generado y las ejecutan. Las acciones lógicas se usan principalmente para modificar variables del juego, de las cuales dependen otras cosas como la condicionalidad de los diálogos o las posibles acciones que tiene Ego durante la aventura.

Por ejemplo, si se quiere hacer que un actor aparezca en una escena en función de la posesión de un objeto por parte de Ego, se puede crear una acción lógica que escucha al evento “entrar en escena”. Lo primero que hará la acción lógica es verificar si el evento ha sido generado por la escena deseada. Luego consultará el valor de la variable de posesión del objeto para saber si Ego lo ha recogido en una escena previa. Si es así, modificará la

 <p>Universidad de Deusto Facultad de Ingeniería</p>	<h2>4 Tipos de recursos</h2>	<p>Proyecto Fin de Carrera</p>
---	------------------------------	--------------------------------

variable de visibilidad del actor deseado. En el caso de que alguno de estas condiciones falle, la variable de visibilidad del actor será puesta a cero, con lo que no aparecerá en escena.

Las acciones lógicas no tienen ningún atributo, únicamente contienen la secuencia de pasos lógicos en un formato abstracto que está descrito en el capítulo 6. Al intentar visualizar mediante doble click el recurso, se abrirá la ventana de módulo lógico con el recurso seleccionado.

#### 4.2.4 Líneas temporales


Las líneas temporales sirven para expresar la secuencialidad de un número de eventos en el tiempo. Son elementos completamente abstractos que por sí solos no tienen valor representativo para el usuario y no se pueden seleccionar desde el gestor de recursos. No obstante, son usados por otros recursos. Una línea temporal es una simple sucesión de los siguientes valores, que a partir de ahora se llamarán “frames”:

- Duración en segundos de este frame.
- Lista de referencias a otros recursos. La lista se plantea a nivel abstracto, y puede contener un número ilimitado de recursos, siempre y cuando todos ellos sean o bien sprites o bien sonidos. Tampoco hay límite en cuanto al número de recursos repetidos del mismo tipo que pueda haber. En el caso de sonidos, la solución es sencilla, simplemente se reproducirán todos a la vez. Quizás sea una paradoja en el caso de que un frame tenga asociados dos sprites, ya que los elementos en las aventuras gráficas son siempre representados de forma individual. Pero esto será tratado por el plugin exportador, de tal forma que quizás sea útil: en función de un parámetro del plugin, éste podría generar una aventura con uno u otro sprite, de forma similar al modo de “compilación optimizada” y “compilación para depurar” que existen en un compilador de C.

#### 4.2.5 Animaciones

Las animaciones son un contenedor especial que únicamente consta de los siguientes elementos:

- Línea temporal asociada, la cual hace referencia a otros elementos como sprites o sonidos. Si la línea hace referencia a otra animación, se permitirá el caso, pero será el plugin exportador quien trate esta incoherencia, posiblemente ignorando cualquier animación anidada.
- Indicador de bucle. Si este valor está activado, la animación será cíclica. De lo contrario, la animación seguirá la línea de tiempo y llegado a su fin, la representación de la animación permanecerá con el último sprite referido por la línea temporal.
- Posición X, Y, Z. Funciona exactamente igual que los valores X, Y, Z de un sprite simple.
- Color RGB de representación lógica. Cuando Tennacles no puede representar la animación en pantalla usando el primer sprite referenciado por la línea temporal, usará un recuadro translúcido del color indicado, que por defecto será azul.

 <p>Universidad de Deusto Facultad de Ingeniería</p>	<p>4.2 Recursos lógicos</p>	<p>Proyecto Fin de Carrera</p>
---	-----------------------------	--------------------------------

Para representar el recurso en una escena, Tennacles buscará el primer gráfico referenciado por la línea temporal y usará sus valores de sprite para posicionarlo en la pantalla. Esto crea un conflicto lógico en el hecho de que tanto el sprite animado como el sprite individual referenciado tienen ambas coordenadas X, Y, Z.

Este conflicto se resuelve de la siguiente manera: Tennacles usará como posición lógica del primer sprite la posición lógica asignada de la animación. Para sucesivos frames con gráfico asociado, Tennacles cogerá la posición lógica del nuevo gráfico y calculará la diferencia de posición respecto al primer gráfico de la línea temporal. Una vez calculado, el gráfico será representado usando la posición de la animación tras aplicar la diferencia calculada.

Esto permitirá al usuario reposicionar las animaciones sin tener que modificar las coordenadas lógicas de los sprites individuales referenciados por la línea temporal asociada, ya que éstas funcionarán de forma relativa al origen de la animación, marcado por la posición lógica del primer sprite referenciado.

En el caso de que una animación tenga una línea temporal asociada que no haga referencia a ningún elemento gráfico, Tennacles dibujará un cuadrado opaco translúcido cuya arista tendrá la longitud de un sexto de la anchura de la pantalla física para que sea manejable desde el editor de escenas como el resto de los sprites. El color puede ser elegido por el usuario.

#### 4.2.6 Actores

Los actores son los elementos más complejos de una aventura, pues se involucran en la lógica del juego. Aunque es posible crear una animación que parezca móvil en la pantalla, el usuario no podrá más que verla. En cambio, un actor es un elemento de interacción: el usuario podrá señalarlo, manipularlo si se trata de un objeto o hablar con él si se trata de un actor con diálogo asignado. Un actor también puede ser un elemento que no interactúa directamente con el usuario, pero que en función de su actuación u otros eventos, modifica su estado.


Por lo tanto no hay diferencia a nivel lógico entre los objetos y personajes que ve un jugador, por la simple razón de que sería posible en una aventura fantástica que un personaje funcionase a veces como objeto (un pájaro parlanchín, un baúl con personalidad, etc). Los atributos comunes a todos los actores son:

##### Visibilidad

Los actores no tienen una relación física con una escena concreta al igual que Ego tampoco la tiene. Los actores siempre están disponibles para aparecer en la escena actual, y será este atributo el que permitirá su aparición o no. Este atributo tendrá que ser modificado por una acción lógica activada por un evento producido al entrar o salir de una habitación: para simular que un actor pertenece siempre a una escena concreta, el evento de entrada en la escena por parte de Ego desencadenará una acción lógica que activará este valor booleano. En cuanto Ego salga de la escena, el evento de salida pondrá este valor a cero para que el actor no continúe “vivo” en la siguiente escena.

##### Animaciones

Se trata de las animaciones que están asociadas con el desplazamiento básico del objeto, la acción de uso y la acción de habla. Cada tipo de animación tendrá asociada tupla de 8 referencias a animaciones que representan el tipo de animación

 <p>Universidad de Deusto Facultad de Ingeniería</p>	<h2>4 Tipos de recursos</h2>	<p>Proyecto Fin de Carrera</p>
---	------------------------------	--------------------------------

mirando en una de las 8 posibles direcciones del actor (norte, sur, este, oeste y las correspondientes diagonales). Cualquier referencia puede ser nula.

Las animaciones de desplazamiento serán visualizadas de forma automática cuando la posición lógica del actor sea modificada. Las animaciones de uso se activan cuando Ego interacciona con el objeto. Este tipo de animaciones pueden representar a Ego agachándose o haciendo ademán de coger algo. Las animaciones de habla serán mostradas cuando Ego entable un diálogo con otro actor. En este tipo de situaciones, se mostrará el primer frame de la animación de forma estática mientras el actor permanezca callado, y cuando tenga algo que decir se pondrá en marcha la animación en un bucle hasta que termine de hablar.

### Acciones lógicas

Los actores reaccionan ante los siguientes eventos que genera Ego: examinar, usar y hablar. Para que cualquiera de estas acciones funcione, el actor debe tener una acción lógica asociada. En el caso de examinar, la acción lógica podría simplemente hacer que Ego exclame una descripción del elemento. En el caso de usar, Ego podría indicar que no puede accionar el objeto, o que no tiene nada que dar al personaje con el que dialoga. La acción de hablar puede lanzar un diálogo. Por supuesto las acciones asociadas pueden ser mucho más complejas. En el caso de que una acción lógica no esté asociada Ego no podrá interactuar con el actor. Es decir, si no hay acción lógica asociada al habla, durante el juego el usuario no podrá hablar con el actor.

#### 4.2.7 Escenas

Las escenas son el contenedor lógico más importante de todos, pues representa la agrupación de recursos que el usuario verá en el juego final. Una escena contiene otros elementos lógicos, pero ningún elemento lógico puede contener una escena. Sólo una escena puede hacer referencia a otra. Al margen de esta distinción, una escena es un contenedor lógico como cualquier otro.

Una escena contiene una lista de capas, y cada capa contiene una lista de recursos. Las primeras tres capas (capa de fondo, capa de Z-buffer y capa de desplazamiento) almacenan únicamente recursos de tipo sprite. La cuarta capa (zonas calientes) almacenará tuplas de cinco elementos: la posición X, Y de la zona caliente, su anchura y altura, y una referencia a otra escena. La última capa contiene los recursos de tipo actor.

 <p>Universidad de Deusto • • • • Facultad de Ingeniería</p>	5 Otras clases	Proyecto Fin de Carrera
---	----------------	----------------------------

## CAPÍTULO 5

# Otras clases

Aunque los recursos son los objetos principales del programa, cuya única función es manejarlos, hacen falta otras clases y objetos no visibles de cara al usuario para que la aplicación funcione correctamente.

### 5.1 Proyecto

---

La clase Proyecto es la que engloba todos los recursos y en definitiva la aplicación a nivel lógico. La ejecución del programa comienza cuando el usuario elige una opción en la ventana de bienvenida, porque es entonces cuando se conocen los parámetros necesarios para la construcción de la clase, como el directorio donde se va a crear el repositorio.

La clase Proyecto contiene una lista con todos los recursos lógicos manejados excepto las escenas, que van en un listado aparte. Ambos tipos de recursos son modificados por sus respectivas ventanas de edición, que hacen de interfaz contra cada recurso individual.

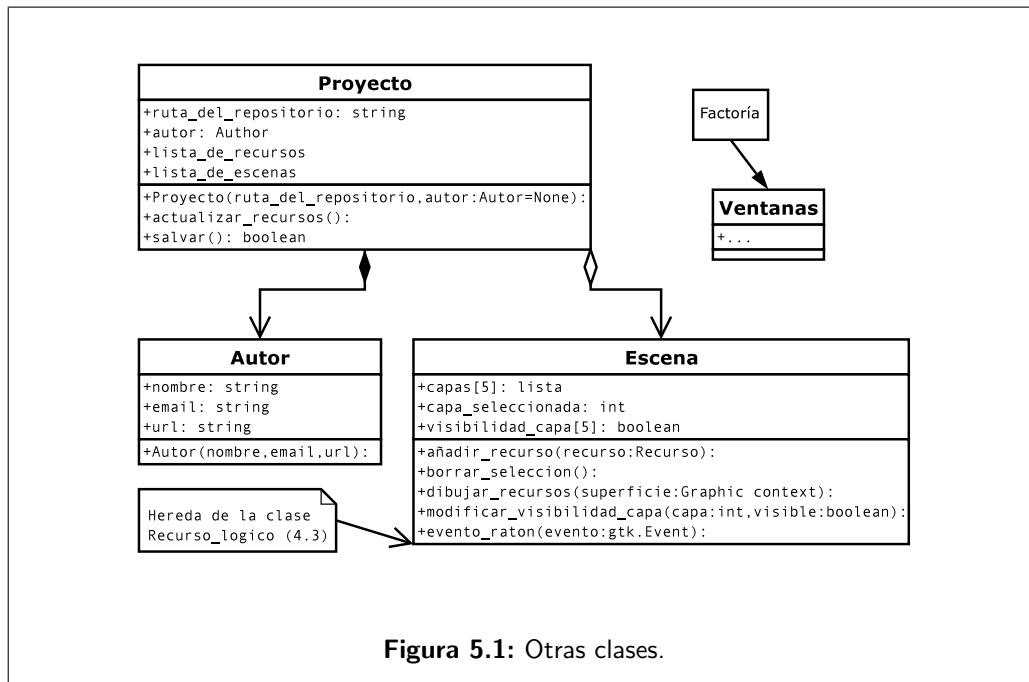
En el plano general, la clase Proyecto implementa el método ‘actualizar\_recursos()’. Este es un método que recorre el disco duro actualizando los recursos, y su algoritmo se explica en el capítulo 7. También se explica ahí el método ‘salvar()’, que recorre la lista de recursos guardando los mismos en el fichero XML del repositorio.

No hay un método de carga, puesto que la carga se realiza de forma implícita con la creación de un recurso. Si durante la construcción existe el fichero de repositorio usado como parámetro, éste será cargado en memoria. De lo contrario, se creará un nuevo repositorio.

### 5.2 Autor

---

La clase autor se usa como un contenedor simple de las cadenas de texto ‘nombre’, ‘email’ y ‘url’, usadas como atributos de una clase Proyecto. No tiene operaciones ni atributos aparte de esos.



### 5.3 Escena

La clase Escena es un recurso de tipo lógico que contiene referencias a otros recursos lógicos. En este aspecto es similar a la clase Animacion, por ejemplo. Pero la clase Escena se encarga de representar los recursos en las ventanas de diálogo. Para realizar esta tarea, la ventana que hace de interfaz contra la clase Escena llamará a ésta cuando el usuario pinche sobre la ventana y/o realice algún movimiento con el ratón. Esto lo hará mediante el método ‘evento\_ratón()’, que recibe como parámetro el tipo de evento de GTK con información sobre el movimiento del ratón o las pulsaciones del teclado.


La clase Escena también ofrece algunos métodos necesarios por su ventana de interfaz, como ‘añadir\_recurso’, usado cuando el usuario arrastra un recurso desde el gestor de recursos hasta la ventana del editor de escenas. ‘dibujar\_recursos’ es una función que se “engancha” en el bucle de proceso de GTK, y es llamada cuando la ventana es desplazada o algo se mueve por encima y tiene que redibujar su contenido. La clase tiene todos los datos necesarios (recursos) excepto la referencia al contexto gráfico de la ventana, que es pasado como único parámetro. El contexto gráfico es el objeto de PyGTK que permite dibujar bitmaps en la ventana.

‘borrar\_seleccion()’ es el método asociado a la tecla de borrar, y ‘modificar\_visibilidad\_capa()’ es el método llamado cuando el usuario quiere alterar la opacidad de la capa seleccionada. Los métodos complejos como ‘dibujar\_recursos()’ y ‘evento\_ratón()’ son descritos en el capítulo ??

### 5.4 Ventanas de interfaz

Todas las ventanas de la interfaz que maneja el usuario están implementadas como clases que heredan del objeto ‘gtk.Window’ de PyGTK. Su única misión consiste en construir los elementos (crear botones, listas y luego asignarlas y colocarlas en el sitio adecuado) e



 <p>Universidad de Deusto • • • • Facultad de Ingeniería</p>	<h2>5.4 Ventanas de interfaz</h2>	<p>Proyecto Fin de Carrera</p>
---	-----------------------------------	--------------------------------

implementar algunos métodos que llaman o modifican las clases con las que están asociadas.


La única particularidad de las ventanas es que no tienen “padre”. Todas las ventanas se construyen a través de una función global que hace de factoría. Esta función global puede llamarse desde cualquier sitio de la aplicación y recibe como parámetros el nombre de la ventana que se desea crear en una cadena de texto y cualquier otro parámetro adicional que necesite su constructor.

Por ejemplo, cuando el usuario arranca Tennacles, lo primero que se ejecuta es un pequeño código que llama a la función factoría para crear la ventana de bienvenida. Cuando el usuario selecciona una opción de ésta, por ejemplo la carga de un repositorio, lo que hace la ventana de bienvenida es llamar a la función factoría, que devuelve el nuevo objeto de la ventana.

La función factoría aparte de crear las ventanas se encarga de llevar un listado de las que están activas y con qué parámetros se crearon. De esta forma la factoría puede evitar la construcción de dos ventanas con el mismo contenido. De hecho, cuando la ventana ya existe, lo que hace es invocar su método ‘raise()’, heredado de la clase ‘gtk.Window’ que lo que hace es “elevar” la ventana por encima de las demás para que sea visible.

La aplicación se acaba cuando se cierran todas las ventanas, pero esto no está implementado por la función factoría sino por PyGTK. Cada ventana creada llama al método ‘gtk.main()’, que crea el hilo encargado de procesar los eventos de la ventana. La destrucción de una ventana llama al método ‘destroy()’ de la ventana. Es PyGTK quien lleva el recuento de llamadas que se hacen a ‘main()’ o ‘destroy()’, y cuando la cuenta llega a cero, libera todos los recursos del sistema finalizando la aplicación porque ya no hay más hilos ejecutándose.



 <p>Universidad de Deusto Facultad de Ingeniería</p>	<p>6 Diccionario de datos</p>	<p>Proyecto Fin de Carrera</p>
--	-------------------------------	--------------------------------

## CAPÍTULO 6

# Diccionario de datos

Este capítulo describe el formato con el cual se almacenarán los datos almacenados en el repositorio de la aventura gráfica. Dado que el fichero de repositorio se almacena en formato XML, se muestra su fichero DTD. El fichero XML funcionará básicamente como una base de datos jerárquica. A continuación se muestran los extractos del DTD que definen los nodos que contienen la persistencia de los recursos. Al final del capítulo se muestra un ejemplo completo de cómo quedaría un fichero XML de repositorio con al menos un ejemplo de todos los posibles nodos.

### 6.1 Nodo raiz

---

```
<!ELEMENT repositorio_tennacles (autor,recursos)>
```

### 6.2 Nodo autor

---

```
<!ELEMENT autor (nombre, email, url)>
<!ELEMENT nombre (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT url (#PCDATA)>
```

### 6.3 Nodo recursos

---

```
<!ELEMENT recursos (imagen | audio | texto | sprite | sonido |
accion_logica | linea_temporal | actor | animacion | escena)*>
```



## 6.4 Nodo imagen

```
<!ELEMENT imagen (ruta, descripcion, tamaño, fecha_modificacion,  
                anchura, altura, bpp)>  
<!ELEMENT ruta (#PCDATA)>  
<!ELEMENT descripcion (#PCDATA)>  
<!ELEMENT tamaño (#PCDATA)>  
<!ELEMENT fecha_modificacion (#PCDATA)>  
<!ELEMENT anchura (#PCDATA)>  
<!ELEMENT altura (#PCDATA)>  
<!ELEMENT bpp (#PCDATA)>
```

## 6.5 Nodo audio


```
<!ELEMENT audio (ruta, descripcion, tamaño, fecha_modificacion,  
                duracion, samples, frecuencia, bps, canales)>  
<!ELEMENT ruta (#PCDATA)>  
<!ELEMENT descripcion (#PCDATA)>  
<!ELEMENT tamaño (#PCDATA)>  
<!ELEMENT fecha_modificacion (#PCDATA)>  
<!ELEMENT duracion (#PCDATA)>  
<!ELEMENT samples (#PCDATA)>  
<!ELEMENT frecuencia (#PCDATA)>  
<!ELEMENT bps (#PCDATA)>  
<!ELEMENT canales (#PCDATA)>
```

## 6.6 Nodo texto

```
<!ELEMENT texto (ruta, descripcion, tamaño, fecha_modificacion)>  
<!ELEMENT ruta (#PCDATA)>  
<!ELEMENT descripcion (#PCDATA)>  
<!ELEMENT tamaño (#PCDATA)>  
<!ELEMENT fecha_modificacion (#PCDATA)>
```

## 6.7 Nodo sprite

```
<!ELEMENT sprite (ruta, descripcion, posicion_x, posicion_y, posicion_z,  
                punto_caliente_x, punto_caliente_y, referencia)>  
<!ATTLIST sprite escalable (true|false) #REQUIRED>  
<!ELEMENT ruta (#PCDATA)>  
<!ELEMENT descripcion (#PCDATA)>  
<!ELEMENT posicion_x (#PCDATA)>  
<!ELEMENT posicion_y (#PCDATA)>  
<!ELEMENT posicion_z (#PCDATA)>  
<!ELEMENT punto_caliente_x (#PCDATA)>  
<!ELEMENT punto_caliente_y (#PCDATA)>  
<!ELEMENT referencia (#PCDATA)>
```

 <p>Universidad de Deusto Facultad de Ingeniería</p>	<p>6.8 Nodo sonido</p>	<p>Proyecto Fin de Carrera</p>
--	------------------------	--------------------------------

## 6.8 Nodo sonido

```
<!ELEMENT sonido (ruta, descripcion, volumen+, duracion, referencia)>
<!ELEMENT ruta (#PCDATA)>
<!ELEMENT descripcion (#PCDATA)>
<!ELEMENT volumen (#PCDATA)>
<!ATTLIST volumen num NMTOKEN #REQUIRED>
<!ELEMENT duracion (#PCDATA)>
<!ELEMENT referencia (#PCDATA)>
```

## 6.9 Nodo accion\_logica

```
<!ELEMENT accion_logica (ruta, descripcion, pasos)>
<!ELEMENT ruta (#PCDATA)>
<!ELEMENT descripcion (#PCDATA)>
<!ELEMENT pasos (#PCDATA)>
```

## 6.10 Nodo linea\_temporal

```
<!ELEMENT accion_logica (ruta, descripcion, frame*)>
<!ELEMENT ruta (#PCDATA)>
<!ELEMENT descripcion (#PCDATA)>
```

## 6.11 Nodo frame

```
<!ELEMENT frame (duracion, referencia*)>
<!ELEMENT duracion (#PCDATA)>
<!ELEMENT referencia (#PCDATA)>
<!ATTLIST referencia num NMTOKEN #REQUIRED>
```

## 6.12 Nodo actor

```
<!ELEMENT actor (ruta, descripcion, animacion_ir, animacion_usar,
animacion_hablar, accion_logica_examinar,
accion_logica_usar, accion_logica_hablar)>
<!ELEMENT ruta (#PCDATA)>
<!ELEMENT descripcion (#PCDATA)>
<!ELEMENT accion_logica_examinar (#PCDATA)>
<!ELEMENT accion_logica_usar (#PCDATA)>
<!ELEMENT accion_logica_hablar (#PCDATA)>
```



### 6.13 Nodo animacion\_ir

---

```
<!ELEMENT animacion_ir (referencia*)>  
<!ELEMENT referencia (#PCDATA)>  
<!ATTLIST referencia num NMTOKEN #REQUIRED>
```

### 6.14 Nodo animacion\_usar

---

```
<!ELEMENT animacion_usar (referencia*)>  
<!ELEMENT referencia (#PCDATA)>  
<!ATTLIST referencia num NMTOKEN #REQUIRED>
```

### 6.15 Nodo animacion\_hablar

---

```
<!ELEMENT animacion_hablar (referencia*)>  
<!ELEMENT referencia (#PCDATA)>  
<!ATTLIST referencia num NMTOKEN #REQUIRED>
```

### 6.16 Nodo animacion


---

```
<!ELEMENT animacion (ruta, descripcion, referencia_linea_temporal,  
                    posicion_x, posicion_y, posicion_z,  
                    color_r, color_g, color_b)>  
<!ELEMENT ruta (#PCDATA)>  
<!ELEMENT descripcion (#PCDATA)>  
<!ELEMENT referencia_linea_temporal (#PCDATA)>  
<!ATTLIST referencia_linea_temporal bucle (true|false) #REQUIRED>  
<!ELEMENT posicion_x (#PCDATA)>  
<!ELEMENT posicion_y (#PCDATA)>  
<!ELEMENT posicion_z (#PCDATA)>  
<!ELEMENT color_r (#PCDATA)>  
<!ELEMENT color_g (#PCDATA)>  
<!ELEMENT color_b (#PCDATA)>
```

### 6.17 Nodo escena

---

```
<!ELEMENT escena (ruta, descripcion, capa*)>  
<!ELEMENT ruta (#PCDATA)>  
<!ELEMENT descripcion (#PCDATA)>
```

 <p>Universidad de Deusto Facultad de Ingeniería</p>	6.18 Nodo capa	Proyecto Fin de Carrera
---	----------------	----------------------------

## 6.18 Nodo capa

---

```
<!ELEMENT capa (referencia*)>  
<!ATTLIST capa num NMTOKEN #REQUIRED>  
<!ELEMENT referencia (#PCDATA)>  
<!ATTLIST referencia num NMTOKEN #REQUIRED>
```



## 6.19 Ejemplo de un fichero XML

```
<?xml version="1.0" encoding="iso-8859-1"?>
<repositorio_tennacles>
  <autor>
    <nombre>...</nombre>
    <email>...</email>
    <url>...</url>
  </autor>
  <recursos>
    <imagen>
      <ruta>...</ruta>
      <descripcion>...</descripcion>
      <tamano>...<tamano>
      <fecha_modificacion>...</fecha_modificacion>
      <anchura>...</anchura>
      <altura>...</altura>
      <bpp>...</bpp>
    </imagen>
    <audio>
      <ruta>...</ruta>
      <descripcion>...</descripcion>
      <tamano>...<tamano>
      <fecha_modificacion>...</fecha_modificacion>
      <duracion>...</duracion>
      <samples>...</samples>
      <frecuencia>...</frecuencia>
      <bps>...</bps>
      <canales>...</canales>
    </audio>
    <texto>
      <ruta>...</ruta>
      <descripcion>...</descripcion>
      <tamano>...<tamano>
      <fecha_modificacion>...</fecha_modificacion>
    </texto>
    <sprite escalable="true">
      <ruta>...</ruta>
      <descripcion>...</descripcion>
      <posicion_x>...</posicion_x>
      <posicion_y>...</posicion_y>
      <posicion_z>...</posicion_z>
      <punto_caliente_x>...<punto_caliente_x>
      <punto_caliente_y>...<punto_caliente_y>
      <referencia>...</referencia>
    </sprite>
    <sonido>
      <ruta>...</ruta>
      <descripcion>...</descripcion>
      <volumen num="x">...</volumen>
      <duracion>...</duracion>
      <referencia>...</referencia>
    </sonido>
  </recursos>
</repositorio_tennacles>
```





```
<accion_logica>
  <ruta>...</ruta>
  <descripcion>...</descripcion>
  <pasos>...</pasos>
</accion_logica>
<linea_temporal>
  <ruta>...</ruta>
  <descripcion>...</descripcion>
  <frame>
    <duracion>...</duracion>
    <referencia num="x">...</referencia>
  </frame>
</linea_temporal>
<actor>
  <ruta>...</ruta>
  <descripcion>...</descripcion>
  <animacion_ir>
    <referencia num="x">...</referencia>
  </animacion_ir>
  <animacion_usar>
    <referencia num="x">...</referencia>
  </animacion_usar>
  <animacion_hablar>
    <referencia num="x">...</referencia>
  </animacion_hablar>
  <accion_logica_examinar>...</accion_logica_examinar>
  <accion_logica_usar>...</accion_logica_usar>
  <accion_logica_hablar>...</accion_logica_hablar>
</actor>
<animacion>
  <ruta>...</ruta>
  <descripcion>...</descripcion>
  <referencia_linea_temporal bucle="true">
    ...
  </referencia_linea_temporal>
  <posicion_x>...</posicion_x>
  <posicion_y>...</posicion_y>
  <posicion_z>...</posicion_z>
  <color_r>...</color_r>
  <color_g>...</color_g>
  <color_b>...</color_b>
</animacion>
<escena>
  <ruta>...</ruta>
  <descripcion>...</descripcion>
  <capa num="x" visible="true">
    <referencia num="x">...</referencia>
  </capa>
</escena>
</recursos>
</repositorio_tennacles>
```



 Universidad de Deusto Facultad de Ingeniería	7 Algoritmos complejos	Proyecto Fin de Carrera
--	------------------------	-------------------------

## CAPÍTULO 7

# Algoritmos complejos

Un programa como Tennacles no tiene apenas algoritmos complejos dado que es prácticamente una interfaz gráfica que ayuda a un usuario a controlar un conjunto de ficheros. No obstante, hay algunas operaciones relacionadas con los mismos que conviene explicar en detalle.

### 7.1 Actualizar recursos

---

La gestión de recursos se realiza desde el método ‘actualizar\_recursos’ de la clase Proyecto. Este método tiene dos objetivos: verificar las fechas de los recursos físicos actualizando sus datos si son posteriores a las fechas almacenadas en memoria, y verificar que los recursos en memoria se corresponden con los disponibles en el disco duro, añadiendo o borrando objetos.

Por lo tanto el método lo que hace es recorrer de forma recursiva todos los ficheros y directorios que existen en el disco duro a partir de la ruta donde se almacena el fichero XML con el contenido del repositorio.

Primero se obtiene de cada directorio la lista de ficheros que existen. Entonces se obtiene una lista temporal con los recursos que actualmente hay en memoria y que se corresponden con ese directorio del disco duro para compararlos. Primero se eliminan de la lista temporal de recursos aquellos objetos cuyos ficheros no existen. Luego se procesa el listado de ficheros intentando crear un recurso físico de cada uno de ellos. El constructor devolverá se comparan con los recursos que se corresponden con ese directorio.

Una vez actualizado el disco duro con los objetos de memoria, se recorren los recursos verificando la fecha de modificación del fichero con el que están relacionados. Si la fecha de modificación del fichero es anterior a la que almacena el recurso en memoria, no ocurre nada. En caso contrario, se actualiza la fecha del objeto y se eliminan los datos que se hubiesen obtenido del mismo.



## 7.2 Salvar repositorio

Para salvar el fichero XML sólo hay que tener varias cosas en cuenta:

1. El fichero XML debe seguir una estructura definida por su DTD.
2. Hace falta añadir “enlaces simbólicos” al fichero para que al recuperar los objetos sus relaciones sean correctas.

En principio no hace falta almacenar la información de los recursos simples, ya que ésta es trivial (apenas un par de atributos), pero al almacenarlos en el fichero XML se evita tener que recuperar los mismos, proceso que puede alargarse bastante para jerarquías de recursos con muchos ficheros.

El seguimiento de la estructura definida por el DTD no es un problema. Se puede realizar de dos formas: convirtiendo la estructura de objetos en memoria a un árbol de nodos XML o escribir el formato de forma manual. La segunda opción es más sencilla de implementar, así que será la que se use.

Finalmente, el problema de los enlaces simbólicos se resuelve mediante las rutas relativas a los ficheros. Mientras que en memoria un recurso puede apuntar a otro recurso, en el fichero XML las referencias se almacenaran como rutas. Al recuperar el objeto del fichero, se buscará un recurso que tenga la ruta indicada y se asignará su referencia.

## 7.3 Dibujar recursos

Para dibujar los recursos gráficos en la ventana, la clase Escena usará “el algoritmo del pintor”. Este algoritmo consiste en pintar todos los elementos que deben aparecer en orden de profundidad mayor a menor. De este modo el algoritmo evita tener que calcular si un pixel debe dibujarse en la ventana o no por estar tapado por otro. Además de respetar el valor Z de todos los recursos, se dibujará también todo según el orden de las capas: primero la de fondo, luego el Z-buffer de fondo, etc.


Es durante el dibujado de recursos cuando se recuperan los datos binarios de los gráficos. Hasta el momento, el editor ha funcionado con representaciones de los elementos físicos pero no ha usado su contenido para nada. A la hora de dibujar el elemento se verifica si existe su atributo ‘datos\_binarios’, el cual es obtenido a través de la interfaz de Allegro con PyGTK.

Si hace falta dibujar un recurso lógico que no tiene datos binarios pues apunta a un recurso físico que ha desaparecido del disco duro, o éste no tiene representación (ejemplo: una animación a la cual todavía no se ha asignado un gráfico) se dibujará un recuadro de color azul que ocupe su lugar.

Aparte de dibujar los recursos en sí, se verificará que el recurso dibujado está seleccionado o no, variable que modifica la rutina que detecta los eventos de interacción del usuario sobre la ventana. Si el recurso está seleccionado, dibujará a su alrededor una especie de marco con unos pequeños rectángulos que facilitan su selección visual mediante el ratón.

## 7.4 Controlando el editor de escenas

Debido a que el editor de escenas usa su propio método para dibujar la ventana, la librería PyGTK es totalmente ajena a lo que ocurra en la ventana, desde su punto de vista no existe ningún elemento en la ventana, solamente un gran área no usada. Para

 <p>Universidad de Deusto      Facultad de Ingeniería</p>	<p>7.4 Controlando el editor de escenas</p>	<p>Proyecto Fin de Carrera</p>
--	---	--------------------------------

conseguir que el usuario pueda desplazar e interactuar con los recursos representados de forma virtual sobre la ventana, hace falta “enganchar” la generación de eventos relacionada con el ratón y el teclado sobre la ventana a rutinas propias.

Por defecto PyGTK se encarga de la gestión de estos eventos, lo cual tiene sentido cuando sólo interesa saber que un botón se ha pulsado y si ha sido con el botón izquierdo o el derecho y los enmascara. Pero gracias al mecanismo de señales (ver capítulo 2.3) podemos obligar a PyGTK a que llame una función propia por cada movimiento en el ratón o teclado. Ésta función propia es similar a la que controla el evento de un botón de la interfaz: recibe como parámetro la ventana que generó el evento y el objeto evento, del cual se puede extraer la información de movimiento del ratón.

Cada vez que la ventana del editor de escenas detecte un movimiento de este tipo, pasará el evento a la clase Escena que lo considerará. La clase Escena verificará los datos del evento y en función de la acción del usuario seleccionará un recurso, lo arrastrará o no hará nada.



 Universidad de Deusto Facultad de Ingeniería	8 Interfaz de Allegro con PyGTK	Proyecto Fin de Carrera
--	---------------------------------	-------------------------

## CAPÍTULO 8

# Interfaz de Allegro con PyGTK

Desde el editor de escenas, el usuario compone cada pantalla del juego final de forma gráfica. Aunque Tennacles no puede ofrecer una visión exacta de cómo se verá el juego final, dado que esto es tarea del motor para el cual será exportada la aventura, es necesario como mínimo una vista preliminar gráfica, y para esto se usará Allegro.

Allegro es una librería para programadores de C/C++ orientada al desarrollo de videojuegos, distribuida libremente, y que funciona en las siguientes plataformas: DOS, Unix (Linux, FreeBSD, Irix, Solaris), Windows, QNX y BeOS (la versión MacOS está en estado alpha). Tiene muchas funciones de gráficos, sonidos, entrada del usuario (teclado, ratón y joystick) y temporizadores. También tiene funciones matemáticas en punto fijo y coma flotante, funciones 3d, funciones para manejar ficheros, ficheros de datos comprimidos y una interfaz gráfica.

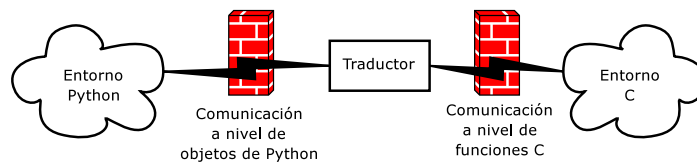
Para la primera versión de Tennacles (ver apéndice 10) sólo se usarán las rutinas de gestión gráfica y sonora para extraer los datos necesarios de los recursos que desea usar el usuario. Dado que Allegro es una librería en C y Tennacles está implementado en Python, es necesario crear una interfaz para que ambos lenguajes se puedan comunicar, lo que comúnmente se llama “binding” o “enlace”.

El enlace entre Python y C se hace a través de un pequeño programa escrito en C (ver figura 8.1), cuya misión es definir qué funciones de C serán visibles desde Python, y la forma en la que se convierten los datos, puesto que para hacer el paso de parámetros de forma correcta, hay que convertir los objetos de Python a C y viceversa, sin menospreciar el hecho de que Python soporta más formas de llamada que C o que C es un lenguaje estático y Python es dinámico.

Lo primero es delimitar qué funcionalidad de Allegro es necesaria y en qué parte del enlace implementarla.

### **Dibujado de recursos en una ventana**

Tennacles usa la librería PyGTK para la interfaz gráfica. Ésta no obstante es un envoltorio de la librería GTK escrita en C. Por lo tanto, existe la posibilidad de



**Figura 8.1:** Esquema de traducción del “Enlace” o “binding” entre los lenguajes Python y C.

“trasladar” los recursos de C a Python y dibujarlos desde PyGTK, o bien implementar un nuevo “widget” de GTK que soporte el dibujo de elementos gráficos a través de Allegro, y hacerlo visible desde Python e integrarlo con PyGTK.

Si consideramos el rendimiento de ambas soluciones, la opción de implementar todo en C y hacerlo visible desde Python es la más óptima. Los recursos los gestiona Allegro, la librería de interfaz gráfica está implementada en C, y el propio lenguaje impone poca o nula sobrecarga sobre la implementación de los algoritmos. En cambio, si se implementa en Python, en primer lugar hay que transferir los datos de C a Python para poder operar con ellos, y para dibujarlos hay que volver a enviarlos varias capas más abajo hasta la librería en C.

Si consideramos el tiempo de desarrollo necesario para tener un programa que realice su función, lo más lógico es programar todo desde Python. La rapidez de desarrollo en Python es claramente superior a C, más libre de fallos, y más sencilla de mantener. Implementar casi todo en C sería un suicidio, por la simple razón de que PyGTK/GTK no tienen apenas documentación sobre la creación de nuevos elementos de interfaz, sin olvidar lo rudimentario que es el lenguaje C y sus limitaciones en cuanto a tipo de datos, gestión de memoria o tratamiento de errores.

Dadas las características de este proyecto la primera opción es la acertada.

### Gestión de los recursos


Según la opción anterior, solamente hace falta proporcionar un par de funciones a Python para cargar, dibujar y actualizar los recursos, lo cual se puede hacer perfectamente en un par de funciones. De esta manera se puede ahorrar la implementación de un gestor de recursos completo en C, capaz de ordenar los recursos a nivel gráfico para dibujarlos, y un conversor de objetos de Python a C.

Por lo tanto, se describen a continuación el prototipo y algoritmo de las funciones que deben ser visibles desde el entorno de Python:

### Iniciación de Allegro

Antes de poder usar cualquier función de Allegro, la librería debe ser iniciada correctamente o el resto de las funciones fallarán en su ejecución. Debido a la forma dinámica de Python de importar código externo, no hace falta hacer una llamada explícita a la función de iniciación.



 <p>Universidad de Deusto Facultad de Ingeniería</p>	<h2>8 Interfaz de Allegro con PyGTK</h2>	<p>Proyecto Fin de Carrera</p>
---	--	--------------------------------

Todos los módulos tienen una especie de rutina principal que se ejecuta de forma implícita la primera vez que el módulo es importado, y es aquí donde se colocarán las llamadas necesarias a Allegro.

### Información sobre un recurso gráfico

Esta función debe tener un único parámetro que será la ruta del fichero hasta el recurso en el disco duro. Si Allegro es capaz de manejar el fichero indicado, la función deberá devolver una tupla con los siguientes valores numéricos:

1. La anchura del gráfico en pixels.
2. La altura del gráfico en pixels.
3. La profundidad de color del gráfico expresada en bits por pixel. (8, 15, 16, 24 o 32).

En el caso de que Allegro no pueda manejar el fichero indicado (por ejemplo si el fichero está corrompido o Allegro no soporta su formato), la función devolverá el objeto único “None”, que equivale a decir que no se pudo procesar la petición.

### Información sobre un recurso auditivo

Esta función debe tener un único parámetro que será la ruta del fichero hasta el recurso en el disco duro. Si Allegro es capaz de manejar el fichero indicado, la función deberá devolver una tupla con los siguientes valores numéricos:

1. Duración del sonido en segundos (valor en coma flotante).
2. Frecuencia en Hz del sonido (entero).
3. Número de bits con el que fue muestreado el sonido, 8 o 16 (entero).
4. Número de canales que tiene el sonido indicado como un número, que será normalmente uno para sonidos monofónicos y dos para sonidos estereofónicos (entero).


Al igual que la función de manejo de recursos gráficos, si no se puede manejar el recurso sonoro se devolverá el objeto “None”.

### Datos binarios de la imagen

Para dibujar un gráfico bitmap desde PyGTK, es necesario tener sus datos en Python como una cadena de enteros agrupados en grupos de cuatro “RGBP”, siendo “R” el valor de 0 a 255 del pixel rojo, “G” el valor de 0 a 255 del pixel verde, “B” el valor de 0 a 255 del pixel azul y “P” un valor cualquiera usado para alinear los datos a una palabra de 32 bits.

Por lo tanto esta rutina devolverá el contenido del gráfico en una cadena de Python con este formato. Esto implica dos posibles conversiones necesarias. Por un lado, el recurso gráfico podría ser un bitmap de 8 bits por pixel, con lo cual hay que transformarlo del “modo paletizado” a 32 bits o “modo truecolor”. Por otro lado, quizás el usuario tenga su pantalla ajustada a una profundidad de 16 bits por pixel. En este caso, PyGTK transformará los 32 bits en 16 antes de mandar los datos a la tarjeta de vídeo.

Gracias a la conversión interna a un formato neutral de 32 bits, se garantiza la visualización correcta de cualquier tipo de gráfico con cualquier configuración posible de

 <p>Universidad de Deusto Facultad de Ingeniería</p>	<p>8 Interfaz de Allegro con PyGTK</p>	<p>Proyecto Fin de Carrera</p>
---	--	--------------------------------

pantalla. Además, así no hace falta programar rutinas de conversión específicas para cada caso. Si tenemos en cuenta  $N$  posibles profundidades de color de los recursos gráficos y  $M$  posibles profundidades de pantalla, habría que implementar  $N * M$  rutinas de conversión. Usando el formato neutral de 32 bits sólo hace falta implementar  $N + M$  rutinas.


### Datos binarios del sonido

Dado que Tennacles no tiene capacidad de reproducción sonora (ver apéndice 10), no hace falta este tipo de datos de los ficheros auditivos.

### Conversión de formato gráfico

Aunque Tennacles pueda manejar formatos gráficos como JPEG o PNG, quizás el motor final no sea capaz de manejarlos. El plugin exportador siempre puede obtener acceso a los datos binarios de la imagen en el formato neutral de 32 bits y guardarlos en disco como *dsee*. Dado que Allegro permite salvar gráficos, para evitar tener que “reinventar la rueda” en el plugin exportador, esta función permite convertir un fichero de un formato a otro y será accesible por el plugin.

Se requieren tres parámetros, que serán el nombre del fichero original del recurso y el nombre del fichero destino. El formato de ambos se deducirá por la extensión. El tercer parámetro numérico indica la profundidad de color a la que se desea guardar el fichero. La rutina devolverá un valor booleano para indicar el éxito de la conversión. La definición concreta de esta rutina se hace en el capítulo 9.

 <p>Universidad de Deusto Facultad de Ingeniería</p>	<p>9 Interfaz de Tennacles con plugins exportadores</p>	<p>Proyecto Fin de Carrera</p>
---	---	--------------------------------

## CAPÍTULO 9

# Interfaz de Tennacles con plugins exportadores

Los plugins exportadores son pequeños componentes de software que interactúan directamente con Tennacles para obtener los datos necesarios de la aventura y crear los datos específicos de un motor gráfico, haciendo de conversores. La interfaz con los plugins funciona a nivel del lenguaje Python. Un plugin puede estar escrito en otro lenguaje (por ejemplo C), pero la comunicación con Tennacles tendrá que seguir siendo realizada a nivel de Python. Se recomienda la implementación pura en Python, para que así el plugin sea portable a cualquier plataforma y no dependa de librerías que el usuario podría no tener.

Hay muchas maneras de definir la interfaz entre la aplicación y los plugins, cada una con sus ventajas e inconvenientes. Tennacles permite que los plugins funcionen de manera más o menos autónoma, pidiendo un requisito de interfaz mínimo. Esto da flexibilidad y autonomía al plugin, aunque permite por ejemplo que consuma todos los ciclos de la CPU prácticamente bloqueando el ordenador del usuario. Los datos de Tennacles serán exportados a través de objetos proxy que usen el control de acceso a atributos mencionado en la sección 2.2.4, con lo que se impide que un plugin pueda modificar la estructura interna de los objetos.

Para que un plugin sea “visible” desde Tennacles deberá cumplir las especificaciones de la API exportada por los plugins. El cumplimiento de esta API no garantiza que luego a la hora de exportar los datos no haya problemas. El éxito de esta tarea vendrá determinado por lo bien que el plugin implemente las llamadas a la API exportada por Tennacles y por la habilidad del autor del plugin.


### 9.1 API exportada por los plugins

---

Todo plugin deberá implementar las siguientes funciones:

#### **init(configuración)**

Los plugins se cargan y ejecutan en el espacio de memoria de la aplicación. Esta

 <p>Universidad de Deusto Facultad de Ingeniería</p>	<p>9 Interfaz de Tennacles con plugins exportadores</p>	<p>Proyecto Fin de Carrera</p>
---	---	--------------------------------

función es llamada por primera vez antes que cualquier otra del plugin y le permite iniciar las variables que almacenan su estado. ‘configuración’ es el valor anteriormente devuelto por la función ‘serialize’ del propio plugin que hubiese sido almacenado en el fichero del repositorio. Si esta función no había sido llamado antes, el parámetro tendrá el valor ‘None’ para indicar que se llama al plugin por primera vez durante la vida del repositorio actual.

**description(): (nombre, descripción)**

Esta función que no recibe parámetros deberá devolver una tupla con el nombre y la descripción del plugin, ambos como cadenas de texto. Esta función se llama para identificar el plugin de cara al usuario en la ventana de selección de plugin (ver sección 3.4). El nombre es usado a modo de identificador universal. No puede haber más de un plugin con el mismo nombre, en cuyo se ignorarían todos los que lo usen.

**serialize(): datos XML**

Esta función se usa para permitir almacenar el estado del plugin (básicamente la configuración realizada por el usuario) en el fichero XML del proyecto. El valor devuelto debe ser una cadena de texto que represente todos los datos del plugin en un árbol XML completo e independiente. Éste será procesado, y en caso de cumplir los requisitos mínimos (ser una cadena XML válida), se insertará en el repositorio usando como identificador el nombre del plugin.

**export(nodo raíz): (estado, mensaje)**

Se llamará a esta función cuando el usuario seleccione la opción exportar del gestor de recursos. Como único parámetro se pasará un objeto proxy de la clase que almacena el proyecto, para que el plugin pueda recorrer en memoria los recursos usados por el proyecto. Devuelve una tupla con los valores estado, que indica si la exportación tuvo éxito, y mensaje, el texto que será mostrado al usuario tras finalizar la exportación.

## 9.2 API exportada por Tennacles


La API exportada por Tennacles se hace a través del nodo raíz proxy que se entrega al plugin durante la exportación. Sólo a través de esta clase se podrán leer datos y acceder a las funciones indicadas en el capítulo 8.

**proyecto.scenes**

Es la lista de objetos escena que hace referencia la clase raíz del proyecto, la cual tendrá que ser recorrida para poder exportar el juego. No hace falta mostrar las otras listas, como la que almacena los recursos, puesto que son accesibles a través de los elementos objetos usados por las escenas.

**proyecto.convert(recurso, nombre del fichero, bpp): estado**

Esta rutina permite al plugin convertir un recurso almacenado en un fichero a otro tipo, especificado por la extensión del nombre del fichero. El parámetro recurso es un objeto de las clases proxy, del cual Tennacles extraerá el nombre del fichero origen. El nombre del fichero es la ruta donde se salvará el gráfico una vez convertido al formato especificado por la extensión usada. bpp es un entero que indica la profundidad de color a la que se quiere salvar el gráfico. La función devuelve el valor booleano que indica el éxito de la conversión.

 <p>Universidad de Deusto • • • • Facultad de Ingeniería</p>	9.2 API exportada por Tennacles	Proyecto Fin de Carrera
---	---------------------------------	----------------------------

Las clases proxy son exactamente iguales a las descritas en los capítulos 4 y 5, con la salvedad de que al ser objetos proxy, no se puede modificar sus datos.



 <p>Universidad de Deusto Facultad de Ingeniería</p>	<p>10 Mejoras posibles</p>	<p>Proyecto Fin de Carrera</p>
---	----------------------------	--------------------------------

## CAPÍTULO 10


# Mejoras posibles

Tennacles es susceptible de ser mejorado sustancialmente, de hecho, algunas cosas aquí enumeradas quizás hubiesen sido implementadas en caso de haber dispuesto de más tiempo o personal.

- El editor de escenas podría tener una vista que relacione las escenas de forma espacial. Es decir, si una escena conecta con otra, sería bueno ver ambas unidas por un enlace en un mapa general que el usuario pueda usar también para navegar por la aventura.
- De cara al usuario sería conveniente añadir aparte del manual de usuario, un tutorial que explique paso a paso la creación de una aventura simple desde cero con un conjunto de recursos disponibles con el propio programa. Así la curva de aprendizaje de la herramienta sería más suave.
- La librería Allegro usada para la gestión de recursos a bajo nivel es una dependencia un poco grande para una aplicación. Dada la licencia liberal de Allegro, sería mejor de cara al usuario coger el código necesario de Allegro e implementarlo de forma separada en un pequeño módulo que se distribuya con Tennacles. Se reduciría así el tamaño completo de la aplicación.

Otra variante de esto sería implementar una interfaz genérica sobre las librerías de bajo nivel, similar al mecanismo de plugins exportadores. En otras palabras, Allegro podría tener un plugin en Python que hiciese de interfaz, y podría haber otras interfaces para SDL, DirectX y otras librerías. De este modo Tennacles no dependería exclusivamente de una única librería.

- Para ocultar al usuario parte de la dificultad de la programación, no se le permiten manipular directamente variables lógicas, sólo leerlas o modificarlas. Esto es así porque es la gestión de variables una de las cosas que Tennacles debería facilitar.

 <p>Universidad de Deusto Facultad de Ingeniería</p>	<p>10 Mejoras posibles</p>	<p>Proyecto Fin de Carrera</p>
---	----------------------------	--------------------------------

Desafortunadamente Tennacles no proporciona un mecanismo que ayude al usuario a “recordar” los nombres usados en las variables.

Es decir, si el usuario está usando en una acción lógica la variable ‘objeto\_magico\_puerta’, y en un diálogo quiere verificar su valor, pero se confunde al teclearlo, Tennacles ocultará este sutil fallo y creará que se trata de otra nueva variable. Lo ideal sería que el editor permitiese mostrar una relación de las variables creadas y manejadas por todos los elementos lógicos de la aventura, para que así fuese más fácil localizar fallos, o simplemente ver cómo afecta la modificación de una variable lógica a la aventura.

- Cuando se describe el mecanismo de la factoría (ver sección 5.4) también se describe la limitación de tener una ventana “viva” por cada elemento único: no puede haber dos ventanas con la misma escena, actor, o lo que sea. El siguiente paso sería implementar el patrón “observer”, de tal modo que el usuario pueda abrir cuantas ventanas quiera. Estas ventanas actuarán como vistas del recurso que están observando, y los cambios realizados en una se actualizarán de forma inmediata en otros.

Esta mejora no tiene mucha relevancia en el estado actual del proyecto, orientado a un usuario único, y al evitarla se simplifica la implementación. No obstante, el acceso al repositorio es crítico. El usuario podría abrir dos veces la aplicación Tennacles con el mismo repositorio, y si hace cambios en ambos procesos y luego los salva, sólo se mantendrán los cambios realizados por el último proceso que modificó el fichero XML.

En un entorno de trabajo en el que trabajan varias personas esto no tendría mucho sentido, ya que impediría que dos artistas estuviesen trabajando en dos partes separadas de la aventura pero que inevitablemente están en el mismo repositorio.

Aquí tendría sentido soportar un mecanismo como RMI o CORBA mediante Pyro [26] o Fnorb [7]. Al lanzar la aplicación por primera vez, el usuario que abre el programa automáticamente genera un servidor de Tennacles. El siguiente usuario que intente abrir el repositorio detectaría el servidor y simplemente se conectaría a él para recuperar el repositorio y los objetos. Esto evitaría restricciones innecesarias de bloqueos sobre el proyecto y la necesidad de que cada trabajador mantenga su propia copia del repositorio, teniendo que realizar cada cierto tiempo la complicada tarea de mezclar repositorios de forma manual para incorporar todos los cambios en un mismo repositorio.

Gracias a esto, un proyecto grande podría tener a tres artistas trabajando simultáneamente sobre el mismo. Uno podría estar trabajando sobre los diálogos, el segundo podría estar modificando los escenarios y probando nuevas combinaciones, y el tercero podría estar actualizando los gráficos de los actores existentes o añadiendo más recursos. Tennacles incluso podría implementar una especie de sistema de mensajería inmediata para que los usuarios puedan comunicarse para discutir los cambios que realizan en la aventura.

- Tennacles resulta una herramienta demasiado general, como se ve en el hecho de que no hay forma de crear una IGU para la aventura final. Sería posible añadir “ganchos” en la aplicación que los plugins podrían modificar para modificar el comportamiento de Tennacles o añadir nuevas funcionalidades implementadas por el plugin. Además sería vital que los cambios específicos del plugin fuesen almacenados




 <p>Universidad de Deusto      . . . .      Facultad de Ingeniería</p>	<p>10 Mejoras posibles</p>	<p>Proyecto Fin de Carrera</p>
---	----------------------------	--------------------------------

de forma genérica en el fichero de repositorio para evitar al usuario llevar la gestión de aun más ficheros desperdigados por el disco duro.

- Ayuda sensitiva al contexto. Actualmente sólo desde el gestor de recursos se accede a un manual on-line. Idealmente todas las ventanas de la interfaz y todos los menús emergentes tendrían una opción de ayuda, que en lugar de abrir el manual por el índice, lo abrirían justo en la sección relevante según el contexto.
- Aunque se permite la gestión de elementos sonoros y su asociación con acciones lógicas o con animaciones, Tennacles no permite la reproducción de estos recursos. Quizás debería implementarse un limitado motor gráfico que sea capaz de soportar justo las características generales que permite modificar Tennacles, para evitar tener que recompilar la aventura para ver cómo quedan los cambios.



 <p>Universidad de Deusto Facultad de Ingeniería</p>	<p>11 Conclusión</p>	<p>Proyecto Fin de Carrera</p>
---	----------------------	--------------------------------

# CAPÍTULO 11

## Conclusión

El objetivo de Tennacles era proporcionar a un usuario una herramienta que le permitiera construir desde cero una aventura gráfica sin necesidad de saber programar. Aunque Tennacles en el estado actual permite hacer un par de cosas básicas, está muy lejos de las enormes expectativas descritas en el documento de objetivos. Esto era previsible: una persona y el tiempo dispuesto eran claramente insuficientes para la tarea que normalmente se asignaría a un grupo de varios profesionales trabajando a jornada completa durante al menos un año.

Pero Tennacles como proyecto académico tenía como objetivo plantear un reto a su autor:

### **Programación gráfica**

Aunque anteriormente había realizado programas que manejaran gráficos, es la primera vez que el programa “usa ventanas”, ya que todos los trabajos anteriores fueron de MS-DOS y/o Linux en línea de comando.

### **Portabilidad entre plataformas**


Lo más sencillo hubiese sido trabajar únicamente con la plataforma preferida y olvidarse de las demás, sacrificando portabilidad en favor de sencillez de implementación. Por supuesto el lenguaje usado para implementar Tennacles es multiplataforma, pero la mezcla de C+Allegro+Python nunca había sido verificada hasta la fecha.

### **Uso amplio de librerías**

Aunque en la futura vida profesional llegue a trabajar en un equipo, por muy grande que sea siempre habrá que implementar tareas que ya han sido de sobra programadas por otros. Para evitar “reinventar la rueda”, habré de apoyarme en el software existente, libre o cerrado, y este proyecto ha servido como una prueba inicial de contacto con el problema de la integración de productos de tipo variado.

### **Uso de software innovador**

GTK, Python y Allegro son ciertamente herramientas que no se enseñan en la uni-

 <p>Universidad de Deusto Facultad de Ingeniería</p>	<p>11 Conclusión</p>	<p>Proyecto Fin de Carrera</p>
---	----------------------	--------------------------------

versidad. Haber realizado un proyecto con C++, Java, etc, no hubiese representado más que un ejercicio más grande que los realizados hasta la fecha, planteando pocas dudas en cuanto a su diseño e implementación.

### **Colaborar con el software libre**

Cuando el proyecto fue propuesto, PyGTK todavía estaba en desarrollo (fase alpha) y aunque ahora ha madurado un poco, todavía no está finalizado (fase beta). La primera dificultad ha sido encontrarse fallos de la propia librería y la completa ausencia de documentación, obligándome a mirar el código fuente y preguntar en los grupos de noticia. El esfuerzo ha merecido la pena, y ahora aparte de conocer más PyGTK, he colaborado con documentación al proyecto llevado a través de Internet por James Henstridge.

Por estas razones yo me siento satisfecho de haber realizado este proyecto y haber realizado por primera vez un esfuerzo tan grande, que sin duda me prepara para los futuros acontecimientos en mi vida profesional. Espero con impaciencia que sean aun más difíciles e interesantes que éste.

 <p>Universidad de Deusto Facultad de Ingeniería</p>	<p>Acrónimos</p>	<p>Proyecto Fin de Carrera</p>
---	------------------	--------------------------------

# Acrónimos

**API**

Application Program Interface. Interfaz de programación de la aplicación.

**BMP**

Bitmap. Formato de imagen nativo de Windows.

**bpp**

Número de bits por pixel usados por una imagen.

**CORBA**

Common Object Request Broker Architecture (Object Management Group).

**CPU**

Central processing unit. Unidad central de proceso (del ordenador).

**DTD**

Document type definition. Definición de tipo de documento. Fichero auxiliar usado para validar un fichero XML.

**Ego**

Se así la documentación al personaje que manipula el jugador de la aventura gráfica.

**GIMP**

The GNU Image Manipulation Program. Programa de dibujo gráfico similar a Adobe Photoshop.

**GNU**

GNU is not Unix. Proyecto que comenzó en 1984 con el propósito de desarrollar un sistema operativo completo tipo Unix pero de software libre, liderado por Richard Stallman.

**GTK**

The GIMP Toolkit. Librería usada para implementar IGUs multiplataforma.

**HTML**

Hyper Text Markup Language. Lenguaje con el que se describen las páginas web.

 <p>Universidad de Deusto Facultad de Ingeniería</p>	<p>Acrónimos</p>	<p>Proyecto Fin de Carrera</p>
---	------------------	--------------------------------

### **HTTP**

Hyper Text Transfer Protocol. Protocolo de Internet para transferir datos.

### **IGU**

Interfaz gráfica de usuario.

### **IMAP**

Internet Message Access Protocol. Protocolo de Internet para acceder a cuentas de correo.

### **JPEG, JPG**

Joint Photographic Experts Group. Grupo de desarrollo de un formato de almacenamiento con pérdidas de imágenes de 24 bpps.

### **LBM**

Formato de fichero usado por Deluxe Paint.

### **MFC**

Microsoft Foundation Classes.

### **MIDI**

Musical Instrument Digital Interface. Almacena en un fichero las notas de una canción.

### **MP3**

Moving Picture Experts Group Layer-3 Audio. Formato de audio que usa compresión con pérdida.

### **OGG**

Similar al MP3, este formato es una mejora, proporcionando un 25 % más de compresión con el mismo ajuste de calidad. En realidad es un contenedor de compresores, que suele usar Vorbis por dentro.

### **PCX**

Formato de fichero usado por primera vez en el PC Paintbrush.

### **PDA**

Personal Digital Assistant. Asistente digital personal.

### **PNG**

Portable Network Graphics. Formato gráfico desarrollado a raíz del conflicto de patentes sobre el formato GIF.

### **PNJ**

Personaje no jugador

### **POP**

Post Office Protocol. Protocolo de Internet de de email.

### **PyGTK**

Enlace de GTK para el lenguaje Python.

 <p>Universidad de Deusto Facultad de Ingeniería</p>	<p>Acrónimos</p>	<p>Proyecto Fin de Carrera</p>
---	------------------	--------------------------------

**RGB, RGBP**

Red, Green, Blue. Los elementos rojo, verde y azul que componen un pixel. El cuarto elemento cuando está presente suele usarse como relleno para alinear el grupo a 32 bits.

**SMTP**

Post Office Protocol. Protocolo de Internet de email.

**STL**

Standard Template Library. Librería de plantillas estándar para C++.

**TGA**

Formato de fichero Targa.

**UTF-8**

Codificación Unicode para ficheros de texto. Tiene la particularidad de que el texto ASCII de 7 bits se representa sin modificaciones.

**Vorbis**

Vorbis es el nombre de un esquema específico de compresión de audio normalmente usado en ficheros OGG.

**WAV**

Formato de Windows que almacena las características de una onda sonora.

**WMA**

Windows Media Audio. Un formato musical propietario de Microsoft.

**XML**

eXtensible Markup Language. Lenguaje de etiquetas usado para especificar la estructura de los ficheros de forma interoperable.








## Bibliografía (Libros y artículos)


- [Lut96] MARK LUTZ. “Programming Python”. O’Reilly (1996).
- [MA02] ALEX MARTELLI AND DAVID ASCHER, editors. “Python cookbook”. O’Reilly (2002). Libro de recetas de código práctico con Python.
- [McG00] SEAN MCGRATH. “XML processing with Python”. Prentice Hall (2000).
- [Nor90] DONALD A. NORMAN. “The design of everyday things”. (1990). Materia: ergonomía.



 <p>Universidad de Deusto Facultad de Ingeniería</p>	<p>Bibliografía (Otras fuentes documentales)</p>	<p>Proyecto Fin de Carrera</p>
---	--	------------------------------------

## Bibliografía (Otras fuentes documentales)

- [1] Allegro [online]. URL: <http://alleg.sourceforge.net/>.
- [2] Blizzard [online]. URL: <http://www.blizzard.com/>. Compañía de videojuegos.
- [3] Concurrent versioning system [online]. URL: <http://www.cvshome.org/>.
- [4] Gnu/linux debian [online]. URL: <http://www.debian.org/>. Sistema operativo libre.
- [5] DirectX [online]. URL: <http://www.microsoft.com/windows/directx/>. API de programación de videojuegos de Microsoft.
- [6] John Finlay. Pygtk 2.0 tutorial [online]. URL: <http://www.moeraki.com/pygtktutorial/pygtk2tutorial/index.html>.
- [7] Fnorb [online]. URL: <http://www.fnorb.org>. Un ORB para Python conforme a la especificación CORBA 2.0.
- [8] Gimp [online]. URL: <http://www.gimp.org/>. Programa de retoque gráfico.
- [9] Glib [online]. URL: <http://www.gtk.org/>.
- [10] Javier Gonzalez. Allegroogg [online]. URL: <http://nekros.freeshell.org/delirium/almp3.php>.
- [11] Gnu public license [online]. URL: <http://www.gnu.org/licenses/licenses.html#GPL>.
- [12] Gtk 2.0 [online]. URL: <http://www.gtk.org/>.
- [13] hacker ethic [the jargon dictionary] [online]. URL: [http://info.astrian.net/jargon/terms/h/hacker\\_ethic.html](http://info.astrian.net/jargon/terms/h/hacker_ethic.html).
- [14] Dictionary [online]. URL: [http://www.dictionary.com/search?db=\\*&q=hacker](http://www.dictionary.com/search?db=*&q=hacker).
- [15] Shawn Hargreaves. Jugando al juego Open Source [online]. julio 1999. URL: [http://linuxtoday.com/news\\_story.php3?ltsn=1999-07-05-003-10-NW-LF](http://linuxtoday.com/news_story.php3?ltsn=1999-07-05-003-10-NW-LF).

 <p>Universidad de Deusto Facultad de Ingeniería</p>	<p>Bibliografía (Otras fuentes documentales)</p>	<p>Proyecto Fin de Carrera</p>
---	--	--------------------------------

- [16] Haskell [online]. URL: <http://www.haskell.org/>. Lenguaje de programación funcional.
- [17] Nunzio Hayslip and Javier Gonzalez. Mad-project [online]. URL: <http://mad-project.sourceforge.net/>. Un motor de aventuras gráficas.
- [18] James Henstridge. Python gtk binding [online]. URL: <http://www.daa.com.au/~james/pygtk/>.
- [19] Library gnu public license [online]. URL: <http://www.gnu.org/licenses/licenses.html#LGPL>.
- [20] Png: portable network graphics [online]. URL: <http://www.libpng.org/pub/png/>.
- [21] Lillo. Jpgalleg 2.0 [online]. URL: <http://www.cenizasoft.cjb.net/allegro/jpgalleg-2.0.zip>.
- [22] Lisp [online]. URL: <http://www.lisp.org/>. Lenguaje de programación funcional. LISP es una abreviación de LISP Processing.
- [23] Jean loup Gailly and Mark Adler. Zlib 1.1.4 [online]. URL: <http://www.gzip.org/zlib/>.
- [24] Lucasarts [online]. URL: <http://www.lucasarts.com/>.
- [25] Pango [online]. URL: <http://www.pango.org/>.
- [26] Pyro, python remote objects [online]. URL: <http://pyro.sourceforge.net/>. Mecanismo similar a RMI de Java implementado en Python.
- [27] Éxito de Python en la industria [online]. URL: <http://www.python-in-business.org/success/>.
- [28] Eric Steven Raymond. How to become a hacker [online]. URL: <http://catb.org/~esr/faqs/hacker-howto.html>.
- [29] Savannah [online]. URL: <http://savannah.nongnu.org/>.
- [30] Sdl [online]. URL: <http://www.libsdl.org/>. Librería para desarrollar videojuegos multiplataforma.
- [31] Sierra entertainment [online]. URL: <http://www.sierra.com/>. Compañía de videojuegos, antes llamada Sierra On-line.
- [32] Trolltech. Qt development toolkit [online]. URL: <http://www.trolltech.com/products/qt/>.
- [33] Trolltech [online]. URL: <http://www.trolltech.com/>.
- [34] Vorbis [online]. URL: <http://www.vorbis.com>.
- [35] Peter Wang. Loadpng 1.0 [online]. URL: <http://www.alphalink.com.au/~tjaden/loadpng/>.
- [36] Python [online]. URL: <http://www.python.org/>.