ESCUELA SUPERIOR DE INGENIERÍA

INGENIERÍA TÉCNICA EN INFORMÁTICA DE SISTEMAS

EDITOR DE MAPAS BASADO EN TILES

Carlos Villegas Núñez

15 de octubre de 2010

ESCUELA SUPERIOR DE INGENIERÍA

INGENIERO TÉCNICO EN INFORMÁTICA DE SISTEMAS

EDITOR DE MAPAS BASADO EN TILES

- Departamento: Lenguajes y sistemas informáticos
- Director del proyecto: Antonio García Domínguez, Manuel Palomo Duarte
- Autor del proyecto: Carlos Villegas Núñez

Cádiz, 15 de octubre de 2010

Fdo: Carlos Villegas Núñez

Agradecimientos

Deseo dar las gracias a mi padre y a mi hermano por el apoyo que me han brindado durante la realización de este proyecto. A mis amigos, compañeros, y en especial a mis tutores, Antonio García Domínguez y Manuel Palomo Duarte, por la confianza y paciencia que han tenido conmigo y sin la cual nada de esto hubiera sido posible.

Licencia

Convenciones, notación y formato

A continuación se muestran las principales convenciones, notaciones y formatos utilizados a lo largo de todo el documento.

- Si nos referimos a programas o herramientas utilizadas en el desarrollo del proyecto, la notación será:
  Planner.
- Si nos referimos a una orden, o función de un lenguaje, la notación será:
  `sort`.
- Si nos referimos a una orden que ha de usarse en la consola de cualquier sistema GNU/Linux, la notación será:

```
Orden consola
```

- Si nos referimos a un fragmento de código o de ficheros de texto plano, la notación será:

```
1  #include <iostream>
2  using namespace std;
3  int main(){
4    cout <<"Hola mundo"<<endl;
5    return 0;
6  }
```

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Objetivos

Los videojuegos son un medio de entretenimiento que han tenido un gran impacto en la sociedad en las últimas décadas y que mueve miles de millones de euros en la actualidad. Resulta fácil comprender que muchas empresas hayan decidido invertir su talento en esta industria , creando videojuegos y herramientas que las complementen.

Se pretende crear un editor de mapas basado en «tiles»[1] con el fin principal de ayudar a personas que desarrollan videojuegos y que utilizan «tilemaps»[2] para la creación de sus escenarios o mapas.

El editor de mapas permitirá crear mapas de distintos tamaños, cargar otros ya creados, guardarlos o modificarlos. Los tiles se podrán importar al editor de mapas con la opción correspondiente, guardándose, para su posterior utilización o modificación. Luego, una vez importados, podremos colocarlos en las cuadrículas que componen el mapa, teniendo la opción de realizar cualquier tipo de edición sobre ellos. Finalmente, cuando se termine la creación del mapa podremos visualizarlo y comprobarlo, antes de darle uso.

Se desea que la aplicación sea lo más intuitiva y sencilla posible, para que cualquier persona pueda crear su propio mapa.

Además, el editor incluirá una biblioteca para cargar los mapas en memoria con libSDL para facilitar su uso y compatibilidad.

1.2. Organización del documento

El presente documento se divide en ocho capítulos, en los cuales se describirá con detalles cada uno de los pasos seguidos para la realización de este proyecto.

El primero es en el que nos encontramos. Aparte de esta sección, se encuentran los objetivos que se esperan cumplir.

[1] Celdas que contienen una imagen, normalmente de pequeñas dimensiones , y qué unidas entre sí pueden formar una imagen nueva.

[2] Mapas que son creados a través de la unión de un conjunto de tiles.

En el segundo capítulo nos encargaremos de comentar los conceptos básicos del proyecto, así como hablar del tema a tratar y de las tecnologías que hemos usado.

En los siguientes capítulos nos centraremos en cada una de las fases que se llevan a cabo durante la creación de un sistema software. Por tanto, en el tercer apartado abordaremos la etapa de planificación, detallando las tareas realizadas y el tiempo que se ha dedicado a cada una de ellas. Se presentará mediante un Diagrama de Gannt creado a partir de la herramienta libre *Planner*.

En el cuarto capítulo comentaremos la fase de análisis del sistema que hemos tenido que realizar para finalizar el producto.

En el quinto capítulo hablaremos sobre la fase de diseño, definiendo el sistema con todo detalle.

En el sexto capítulo se llevará a cabo la implementación del editor, las técnicas y herramientas usadas, así como las diferentes estrategias que hemos seguido para realizar el proyecto.

En el séptimo capítulo se finalizará el proceso de desarrollo, explicando las diferentes pruebas a las que ha sido sometido el producto para un correcto funcionamiento.

En el octavo capítulo sacaremos las conclusiones a las que se ha llegado después de la finalización del proyecto, todo lo que se ha aprendido y las posibles extensiones del proyecto.

Por último, se incluye una serie de apéndices en los que se encuentran el manual de usuario, el manual de instalación y el manual del desarrollador.

Capítulo 2

Conceptos básicos

2.1. Visión global del producto

El proyecto que presentamos surge como respuesta a la necesidad de crear un editor de mapas basado en tiles con licencia GNU No es Unix (GNU) General Public License (GPL), de forma que pueda ser libremente expansible, ya que bajo estos términos son escasos los editores que existen.

Este editor permite mediante los «tilesmaps» poder almacenar escenarios con grandes proporciones sin pedir como requerimientos mínimos, cantidades enormes de memoria. La técnica de los «tiles» permite construir escenarios utilizando pequeñas áreas rectangulares de distintas texturas que, uniéndolas unas al lado de otras dan la sensación de que todo lo que vemos es un gráfico de una sola pieza. Ésto permite a los programadores disponer de mapas fabricados a partir de matrices que toman diferentes valores dependiendo del tile que se representan en pantalla.

Por tanto, mediante una serie de pequeños esfuerzos cualquier otro programador pueda ampliar los contenidos del editor, haciéndolo más completo a través de nuevas herramientas y funcionalidades. Además de ir añadiendo nuevos tipo de bibliotecas compatibles con los distintos tipos de lenguajes usados para la creación de videojuegos. Es decir, realizar el proyecto lo más escalable posible.

2.2. Antecedentes

La idea de este tipo de editor no es nueva, de hecho, podemos remontarnos hasta los años 80, momento en el que aparecen los primeros juegos con mapas basados en tiles de la mano de Ultimate Play The Game[1] , y junto a ellos, los primeros editores. Juegos que en su día fueron grandes éxitos, como **Civilization** o **SimCity**, destacaban por este rasgo, los tilesmaps ofrecían una gran facilidad ante la creación de enormes escenarios además de permitir a los usuarios crear sus propios niveles.

Lo normal por aquel entonces es que este tipo de juegos trajera consigo su propio editor de mapas personalizado, con lo que evitaba poder ser reutilizado para otros. Es por ello que se comenzaron a implementar este tipo de editores de carácter general, no enfocado a ningún juego en particular. Editores como **Mappy** o **Tile Studio** se hicieron un hueco en los ordenadores de aquellos usuarios que necesitaban crear mapas para el desarrollo de sus propios videojuegos.

Como hemos comentado anteriormente, la gran mayoría de estos editores no poseen versiones libres, suelen ser de uso privativo u ofrecen pequeñas versiones con las funcionalidades limitadas para un uso

[1]Nombre comercial de la empresa Ashby Computers & Graphics Ltd,desarrolladora de software fundada en 1982.

básico y simple. Otros no se encuentran actualizados, y el lenguaje usado para su desarrollo queda obsoleto, resultando complicado encontrar la ayuda suficiente para continuar con el trabajo.

Es por todo esto por lo que pensamos que se trata de un proyecto viable que se ajusta perfectamente a las necesidades descritas.

2.3. Alcance

Se pretende que el editor sea fácil, intuitivo y atractivo de cara al usuario, que ofrezca todas las herramientas necesarias para la creación de mapas propios a través de tiles, desde las tareas básicas de edición hasta la generación de mapas aleatorios mediante el uso de funciones específicas. Todo ello implementado a través del lenguaje C/C++ y de la librería de desarrollo Qt, ambas enormemente reconocidas y populares, haciendo que cualquier persona con noción de informática pueda dedicarse a ampliar el proyecto. Además incluirá una biblioteca para poder trabajar con libSDL, lenguaje ampliamente utilizado hoy en día para el desarrollo de videojuegos y de licencia libre.

Capítulo 3

Planificación

3.1. Incrementos

Para realizar el desarrollo del proyecto se ha decidido que la mejor opción es usar un modelo incremental. La razón principal para escoger este modelo frente a otros reside en que se va a tratar de hacer cada herramienta del editor de la forma más independiente posible, por lo que iremos consiguiendo una versión cada vez más completa hasta llegar a la versión final.

El modelo incremental ofrece una versión funcional al final de cada incremento, la cual podemos probar y a partir de ella planificar las próximas modificaciones, a fin de que nuestro software cubra las necesidades reales.

El primer incremento consta solo de lo requisitos básicos del sistema, seguido de los principales incrementos realizados en el proyecto **ETiles**.

3.1.1. Incremento 1: Requisitos básicos del sistema

En este incremento se planificó el tiempo necesario para la realización del proyecto y para cada una de las tareas que lo componen. Comenzamos con la especificación global del editor, es decir, que es lo que se desea que el editor realice. Además se buscaron las imágenes, iconos, y tiles que se utilizan en el proyecto. Como era necesario, se crearon algunos propios usando el programa GNU Image Manipulation Program (GIMP)[1].

Figura 3.1: Ejemplo de tileset de distintos tipos de agua

[1]Programa de edición de imágenes, bajo licencia libre.

Aparte de estructurar el tiempo, es necesario conocer las herramientas que vamos a utilizar para el desarrollo del proyecto. Dado que la aplicación haría uso de la biblioteca de desarrollo Qt, que desconocía totalmente, fue necesario dedicar tiempo a conocer su funcionamiento. Al igual ocurrió con LaTeX a la hora de desarrollar el documento.

3.1.2. Incremento 2: Creación del mapa

En este incremento nos centramos en crear la base de nuestro proyecto. Para representar los mapas en nuestro editor debemos crear una estructura que permita almacenarlos. En principio nos centraremos sólo en la parte básica, a partir de unas medidas, creamos dicha estructura que guarde los tiles de los que está compuesto el mapa, y que mediante una pequeña interfaz pueda ser visualizado por pantalla. En esta primera versión no se podrá realizar ningún tipo de acción.

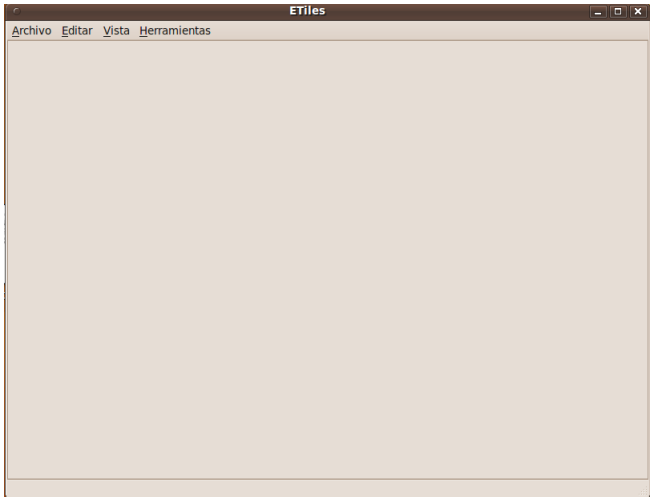

Figura 3.2: Visión del mapa del editor

En la fase de pruebas, creamos un pequeño diálogo mediante el cual el usuario puede introducir las medidas de su mapa, observando en pantalla cómo es creado de forma rápida y correcta para distintos tamaños sin dar ningún tipo de error.

3.1.3. Incremento 3: Creación de la lista de tiles

En esta ocasión nos centramos en la otra parte más importante del editor: la lista de tiles. Como antes, procedemos a la creación de una estructura que almacene los tiles que luego utilizaremos para rellenar y pintar nuestro mapa. Es necesario que esta lista pueda visualizarse de forma gráfica para que el usuario pueda reconocer el tile que desea utilizar en un momento dado de manera inmediata. Podemos conseguirlo si cada tile lo representamos como un icono de unas medidas prefijadas dentro de la lista con una pequeña separación entre cada uno. Además se añaden los iconos de interacción de importación de tile y eliminación de tile.

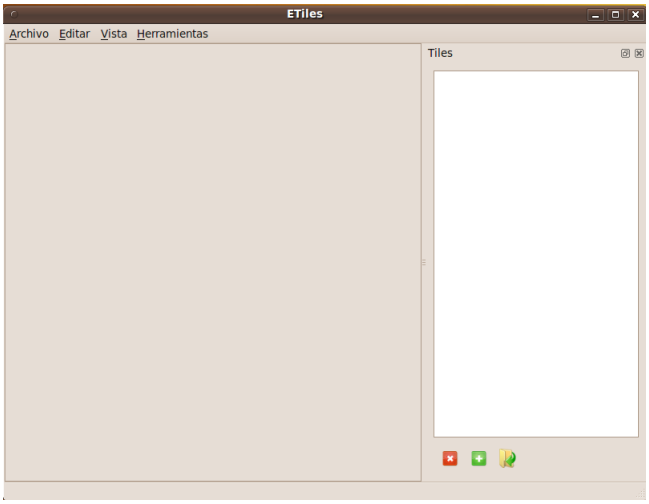

Figura 3.3: Visión del mapa y la lista de tiles

En la fase de pruebas probamos con las distintas imágenes de tiles que disponemos, insertándolas una a una en la lista, y observando como ésta se actualiza de forma correcta, apareciendo el oportuno scroll cuando se excede el tamaño de visión.

3.1.4. Incremento 4: Interacción entre el mapa y la lista de tiles

Una vez realizadas dos de las partes más fundamentales del editor nos enfocamos en la interacción entre ellos. Como sabemos nuestro mapa estará formado por la unión de los tiles que hemos importado previamente a nuestra lista de tiles. Por tanto, debe ser posible seleccionar cada uno de estos tiles de forma individual y que mediante una simple interacción con el ratón se pinte en el lugar del mapa que hayamos indicado.

Ya que hemos conseguido ir rellenando nuestro mapa, aprovechamos también para permitir el desplazamiento entre tiles ya pintados a posiciones vacías del mapa.

Además permitimos que el usuario pueda personalizar el espacio de visión de cada ventana, moviéndola y colocándola en el lugar donde más cómodo se encuentre.

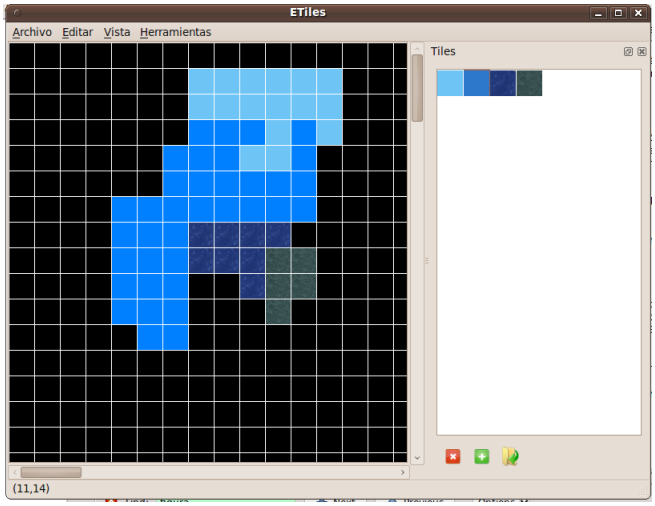

Figura 3.4: Vista de un mapa en desarrollo

Luego realizamos las pruebas pertinentes asegurándonos que el pintado del tile seleccionado se realiza de manera correcta y en la posición indicada por el usuario, al igual que el movimiento.

3.1.5. Incremento 5: Barra de herramientas

A continuación dedicamos nuestros esfuerzos en preparar la barra de herramientas. En ella se encuentran las herramientas más importantes, que serán de vital importancia ya que nos serán de gran utilidad a la hora de desarrollar nuestro mapa. Necesitamos iconos que describan claramente la utilidad que representan para que sean fácilmente identificables por el usuario. Al principio, los iconos no efectuaban ninguna acción y la funcionalidad se añadió a medida que se avanzó de incrementos.

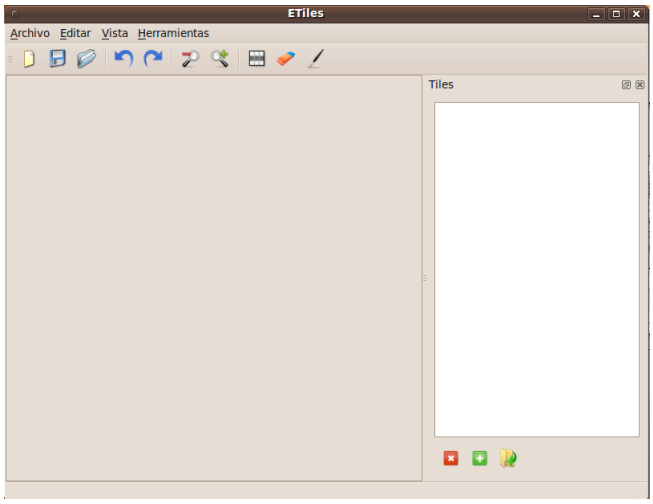

Figura 3.5: Visión de la barra de herramientas

Además retocaremos el menú principal añadiendo enlaces a las acciones que puede realizar la barra de herramientas de manera que pueda accederse desde ambos sitios.

Al realizar las pruebas pertinentes comprobamos que todo se crea de forma correcta.

3.1.6. Incremento 6: Herramienta de relleno y borrado

En este incremento comenzamos con dos de las herramientas que facilitarán mucho la labor de pintado de nuestro mapa. El relleno se dividirá en dos, una primera donde se rellena toda la zona libre del mapa unida al tile donde indique el usuario, es decir, se pinta hasta que encuentre el borde del mapa u otra posición ocupada por un tile. La segunda estará ligada al movimiento del ratón, mientras que lo tengamos pulsado, el mapa se rellena con el tile seleccionado mientras realizamos el desplazamiento del ratón. El borrado es exactamente igual, sólo que se borrará siempre y cuando exista un tile en la posición indicada.

Cuando estén creadas, debemos establecer los enlaces con la barra de herramientas y los menús para poder acceder a ellas.

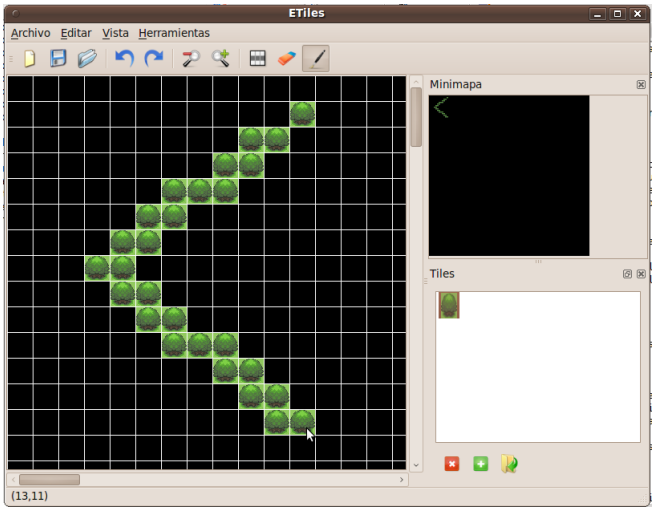

Figura 3.6: Funcionamiento de la herramienta “Rellenar”

Por último realizamos las pruebas asegurándonos que los enlaces funcionan, al igual, que las herramientas. Nos damos cuenta que al realizar el movimiento de arraste del ratón mientras presionamos tenemos que controlar que se realice sobre una posición válida del mapa para que no derive en un error.

3.1.7. Incremento 7: Guardar y cargar

Este incremento es uno de lo más importantes del editor, ya que nos da la posibilidad de guardar y cargar mapas que anteriormente hemos creado y desarrollado. Ahora es el momento de realizar este incremento ya que tenemos las herramientas mínimas para que cualquier usuario pueda completar su mapa de forma sencilla y rápida.

Hay tener especial cuidado al desarrollar el guardado del mapa, ya que la idea es que el formato utilizado pueda ser empleado por las bibliotecas y no surjan problemas de compatibilidad. Por ello, decidimos

crear una función que vuelque el contenido de la matriz de números correspondiente al tile al que representan en un fichero de texto plano, seguido de una línea por cada número de la matriz que identifica un tile.

Aparte del formato descrito, también creamos otra función, existente en todos los editores y también necesaria en éste, que permita almacenar el mapa como una única imagen, ya sea formato Joint Photographic Experts Group (JPEG)[2], BitMaP (BMP)[3], Portable Network Graphics (PNG)[4], etc.

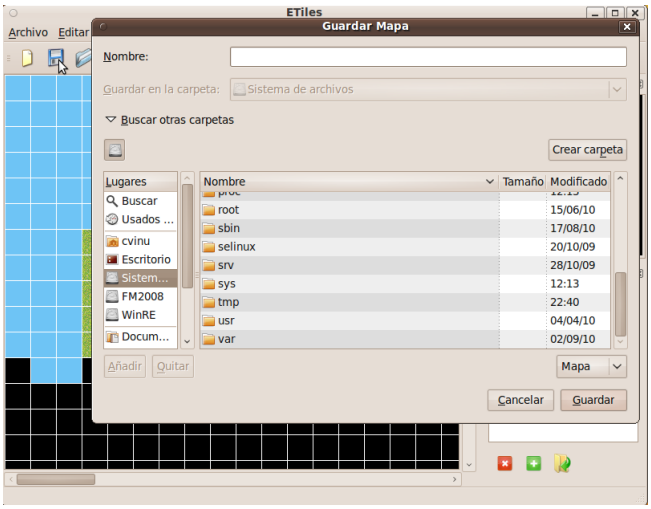

Figura 3.7: Diálogo de la herramienta “Guardar”

Para cargar un mapa seguiremos el proceso inverso al de guardado, leyendo el fichero de texto, rellenando nuevamente la matriz y pintando el mapa. No nos debemos olvidar también de rellenar la lista de tiles.

Para las pruebas comprobamos que el guardado y cargado se producen correctamente.

3.1.8. Incremento 8: Herramientas selección, copiar, cortar y pegar

Después de poder guardar y cargar mapas, volvemos a centrarnos en ampliar las herramientas de edición. En este incremento se implementa la selección, el copiado, el cortado y el pegado de tiles sobre el mapa.

A diferencia de lo que estamos acostumbrados a ver con estas herramientas, en este caso, se comportarán de forma distinta. Y es que hemos pensado que la manera más fácil de poder realizar estas operaciones es a través de rectángulos, es decir, las selecciones serán de forma rectangular. Una vez que tengamos una selección se activarán el resto de opciones. Copiar o cortar sólo surtirá efecto sobre los tiles que existan dentro del rectángulo. Al igual ocurre con pegar, por tanto, en las posiciones vacías del rectángulo

[2] Grupo conjunto de expertos en fotografía, nombre de la comisión que creó la norma, la cual fue integrada desde sus inicios por la fusión de varias agrupaciones en un intento de compartir y desarrollar su experiencia en la digitalización de imágenes. además de ser un método de compresión, es a menudo considerado como un formato de archivo.

[3] es el formato propio del programa Microsoft Paint, que viene con el sistema operativo Windows. Puede guardar imágenes de 24 bits (16,7 millones de colores), 8 bits (256 colores) y menos. Puede darse a estos archivos una compresión sin pérdida de calidad.

[4] Es un formato gráfico basado en un algoritmo de compresión sin pérdida para bitmaps no sujeto a patentes.

no ocurrirá ningún efecto al producirse el pegado.

Por último, se desarrollará un resaltado sobre el rectángulo de selección y pegado para que en todo momento se pueda observar sobre el mapa.

3.1.9. Incremento 9: Herramientas zoom, deshacer y rehacer

En este incremento seguimos aumentando nuestro catálogo de herramientas.

Primero implementamos mejoras en la visión de nuestro mapa, permitiendo a través de la herramienta zoom que los usuarios puedan observar su mapa de diferentes tamaños dependiendo de sus necesidades. Aprovechamos para implementar la rejilla, que no es mas que un conjunto de pequeñas lineas que separan cada tile del mapa vertical y horizontalmente.

A continuación implementamos las herramientas deshacer y rehacer. Ambas sólo serán funcionales en principio sobre el mapa, y no sobre la lista de tiles, permitiendo recuperar cualquier estado anterior.

Durante la fase de prueba todo se comporta de manera correcta, exceptuando que con el zoom apreciamos que con tamaños considerablemente grande de mapas el editor se ralentiza al pintar, por lo nos vemos obligados a modificar el pintado de los mapas en pantalla.

3.1.10. Incremento 10: Herramienta de redimensión y propiedades

En este incremento se añadirán dos nuevas herramientas.

La primera es la herramienta de redimensión, la cual ofrece al usuario la posibilidad de modificar el tamaño de su mapa de forma permanente una vez creado. Ésto puede resultar muy útil ya que en un momento avanzado en la edición si no tuviéramos esta herramienta y nos diéramos cuenta de que nos hemos equivocado a la hora de introducir las medidas o que necesitáramos cambiar el tamaño, perderíamos toda la información ya que tendríamos que crear un nuevo mapa.

La segunda nos proporcionará un pequeño diálogo para cambiar algunas propiedades básicas del editor. En principio se podrán modificar los colores de fondo, de la rejilla, y del rectángulo de selección, ya que cuando estemos editando el mapa, estos colores pueden coincidir con los colores de las imágenes de los tiles produciendo en el usuario confusión. Además se permitirá modificar las posiciones de las ventanas de edición para que el usuario pueda colocarlas a su gusto.

En la fase de pruebas se comprueba que la redimensión se lleva a realizar correctamente, no perdiéndose información. Por otro lado, se cambian las propiedades observando que su modificación es satisfactoria.

3.1.11. Incremento 11: Mapas aleatorios

Terminadas ya todas las herramientas de edición proseguiremos a implementar uno de los aspectos novedosos que hemos decidido incluir en nuestro editor: los mapas aleatorios. Puede ser de enorme utilidad para un usuario crear un mapa completo y coherente mediante una serie de indicaciones simples. Para mantener la coherencia se utilizarán una serie de algoritmos y funciones pseudoaleatorias de forma que cualquier otra persona pueda ampliarlo en cualquier momento para mejorar la aplicación.

En un principio esta opción estará limitada y solo permitirá crear mapas de tierra y agua. El usuario deberá introducir los tiles que componen las dos zonas, además del tamaño del mapa y seleccionar el algoritmo de creación.

Para la fase de pruebas hay que crear un diálogo donde se pueda seleccionar cada una de la opciones citadas anteriormente y asegurarse de que no existan valores nulos en ellas que puedan producir errores a la hora de crear el mapa.

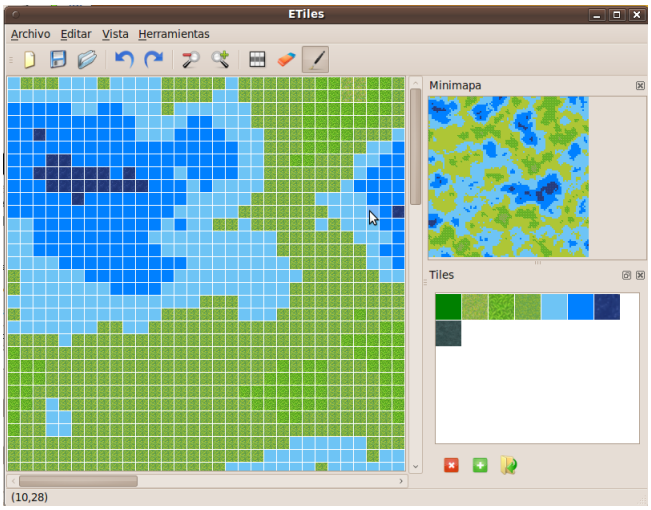

Figura 3.8: Mapa creado con la herramienta “Mapas aleatorios”

3.1.12. Incremento 12: Minimapa

Cuando creamos la función de mapas aleatorios nos damos cuenta que visualizar el mapa completamente mediante un pequeño minimapa puede ser de enorme utilidad para observar los resultados de forma inmediata y no tener que desplazarnos. Es por lo que en este incremento nos centramos en esta utilidad.

El minimapa se mostrará en la parte superior derecha del editor actualizándose con cada paso que damos sobre el mapa. Además implementamos la opción de desplazamiento rápido sobre él, es decir, que al pulsar sobre cualquier posición del minimapa, la ventana de visión del mapa se trasladará a la posición indicada.

Con este penúltimo incremento hemos conseguido terminar con las todas las funcionalidades gráficas del editor.

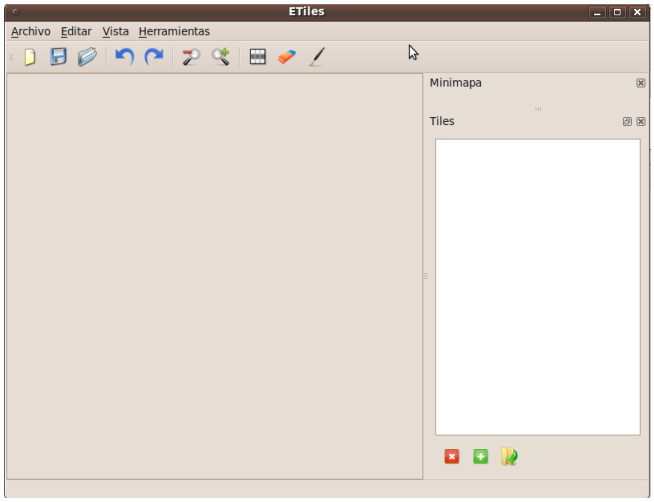

Figura 3.9: Visión del editor completo

Para la fase prueba se comprobaba que al producirse cualquier modificación en la ventana de edición del mapa, inmediatamente se actualiza el minimapa. Además se observaba que el traslado de visión cuando se cliqueaba sobre el minimapa se producía de forma correcta.

3.1.13. Incremento 13: Biblioteca libSDL

Este incremento supone el último gran esfuerzo de código. Se crea una biblioteca que trabaje con libSDL y que permite acceder a cada una de las funciones que proporciona ETiles, de forma que cualquier programador pueda acceder a ellas de forma inmediata sólo añadiendo la biblioteca a su proyecto.

En la fase de pruebas se realizaban pequeños programas de prueba que producían llamadas a las funciones y se comprobaban que su funcionamiento era el correcto.

3.2. Diagrama de Gantt

A continuación se muestran las tareas incluidas en el diagrama de Gannt, realizado con el programa *Planner* donde se observa la planificación de cada uno de los incrementos con sus respectivas tareas. El diagrama de Gannt al completo se adjunta en la carpeta **doc** del proyecto.

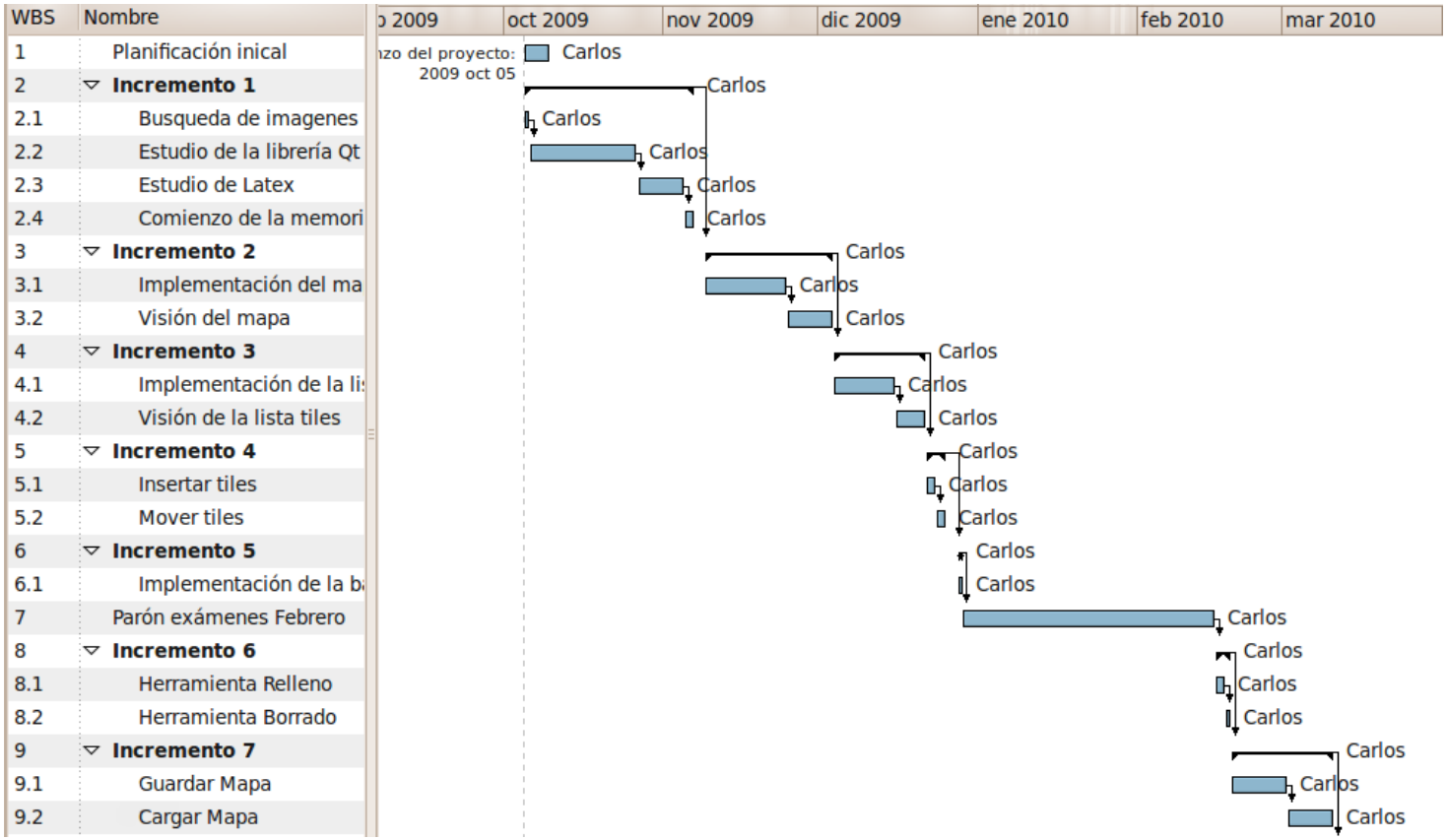

Figura 3.10: 1ª Parte del diagrama de Gantt

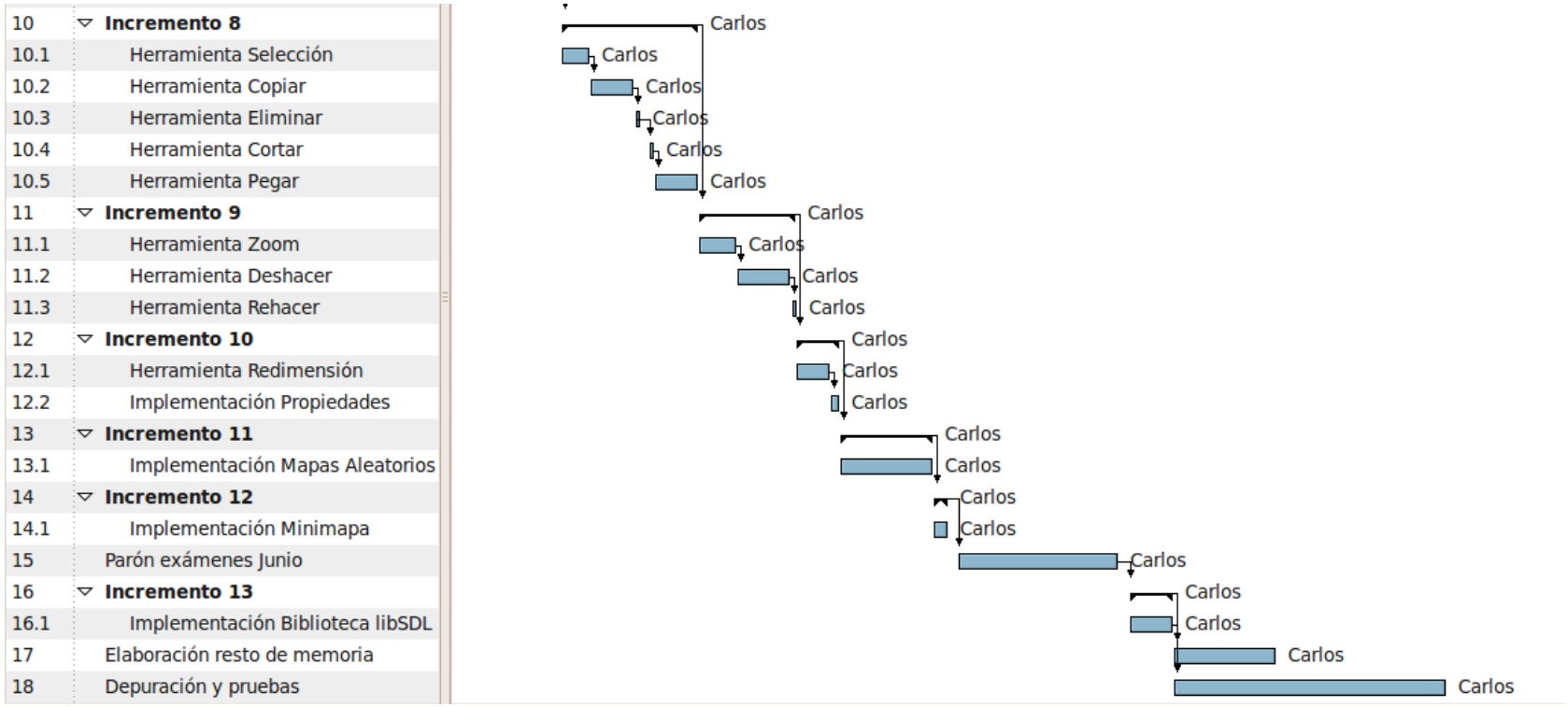

Figura 3.11: 2ª Parte del diagrama de Gantt

WBS	Nombre	Inicio	Fin	Trabajo
1	Planificación inical	oct 5	oct 9	5d
2	**Incremento 1**	**oct 5**	**nov 6**	**25d**
2.1	Busqueda de imagenes	oct 5	oct 5	1d
2.2	Estudio de la librería Qt	oct 6	oct 26	15d
2.3	Estudio de Latex	oct 27	nov 4	7d
2.4	Comienzo de la memoria	nov 5	nov 6	2d
3	**Incremento 2**	**nov 9**	**dic 3**	**19d**
3.1	Implementación del mapa	nov 9	nov 24	12d
3.2	Visión del mapa	nov 25	dic 3	7d
4	**Incremento 3**	**dic 4**	**dic 21**	**12d**
4.1	Implementación de la lista tiles	dic 4	dic 15	8d
4.2	Visión de la lista tiles	dic 16	dic 21	4d
5	**Incremento 4**	**dic 22**	**dic 25**	**4d**
5.1	Insertar tiles	dic 22	dic 23	2d
5.2	Mover tiles	dic 24	dic 25	2d
6	**Incremento 5**	**dic 28**	**dic 28**	**1d**
6.1	Implementación de la barra de herramientas	dic 28	dic 28	1d
7	Parón exámenes Febrero	dic 29	feb 15	35d
8	**Incremento 6**	**feb 16**	**feb 18**	**3d**
8.1	Herramienta Relleno	feb 16	feb 17	2d
8.2	Herramienta Borrado	feb 18	feb 18	1d
9	**Incremento 7**	**feb 19**	**mar 10**	**14d**
9.1	Guardar Mapa	feb 19	mar 1	7d
9.2	Cargar Mapa	mar 2	mar 10	7d
10	**Incremento 8**	**mar 11**	**abr 7**	**20d**
10.1	Herramienta Selección	mar 11	mar 16	4d
10.2	Herramienta Copiar	mar 17	mar 25	7d
10.3	Herramienta Eliminar	mar 26	mar 26	1d
10.4	Herramienta Cortar	mar 29	mar 29	1d
10.5	Herramienta Pegar	mar 30	abr 7	7d
11	**Incremento 9**	**abr 8**	**abr 27**	**14d**
11.1	Herramienta Zoom	abr 8	abr 15	6d
11.2	Herramienta Deshacer	abr 16	abr 26	7d
11.3	Herramienta Rehacer	abr 27	abr 27	1d
12	**Incremento 10**	**abr 28**	**may 6**	**7d**
12.1	Herramienta Redimensión	abr 28	may 4	5d
12.2	Implementación Propiedades	may 5	may 6	2d
13	**Incremento 11**	**may 7**	**may 25**	**13d**
13.1	Implementación Mapas Aleatorios	may 7	may 25	13d
14	**Incremento 12**	**may 26**	**may 28**	**3d**
14.1	Implementación Minimapa	may 26	may 28	3d
15	Parón exámenes Junio	may 31	jul 2	25d
16	**Incremento 13**	**jul 5**	**jul 13**	**7d**
16.1	Implementación Biblioteca libSDL	jul 5	jul 13	7d
17	Elaboración resto de memoria	jul 14	ago 3	15d
18	Depuración y pruebas	jul 14	sep 7	40d

Tabla 3.1: Tabla de tareas incluidas en el diagrama de Gantt

3.3. Esfuerzo

Primero destacaremos que una parte del tiempo se ha dedicado a aprender a usar los distintos lenguajes que han sido necesarios para desarrollar el proyecto, ya sea LaTeX para el documento o la librería de desarrollo Qt para el editor.

Luego destacar que se ha hecho un gran esfuerzo para que la aplicación sea lo más rápida y eficiente posible, ya que para mapas considerablemente grandes se reducía bastante la velocidad. Ésta es una de las tareas que ha provocado que el desarrollo del proyecto no haya cumplido las expectativas de tiempo propuestas en un principio, ya que nos encontramos con problemas ante este tipo de mapas, lo que nos obligó a tener que realizar modificaciones en la implementación hasta encontrar con la más óptima.

Otra de las tareas que cabe mencionar por su esfuerzo es la creación de mapas aleatorios, ya que resulta muy complicado a partir de una serie de indicaciones conseguir que un mapa resulte suficientemente coherente.

Capítulo 4

Análisis

4.1. Metodología de desarrollo

En este proyecto se ha seguido el enfoque que promueve una metodología de desarrollo ágil de software[1], más concretamente, la programación extrema o eXtreme Programming (XP), debido a la simplicidad de sus reglas y prácticas, su orientación a equipos de desarrollo de pequeño tamaño, su flexibilidad ante los cambios y su ideología de colaboración. La diferencia inmediata con otras metodologías es que son menos orientados al documento, exigiendo una cantidad más pequeña de documentación para una tarea dada. De muchas maneras son más bien orientados al código: siguiendo un camino que dice que la parte importante de la documentación es el código fuente[2].

La XP considera que los cambios de requisitos sobre la marcha son un aspecto natural, inevitable e incluso deseable del desarrollo de proyectos. Creen que ser capaz de adaptarse a los cambios de requisitos en cualquier punto de la vida del proyecto es una aproximación mejor y más realista que intentar definir todos los requisitos al comienzo del proyecto e invertir esfuerzos después en controlar los cambios en los requisitos. Además, promueve iteraciones en el desarrollo a lo largo de la vida del proyecto, minimizamos los riesgos desarrollando software en cortos períodos de tiempo, estos períodos de tiempo se llaman iteraciones, la cual puede durar de una a cuatro semanas. Cada iteración debe de contener: planificación, análisis de requerimientos, diseño, codificación, revisión y documentación. Una iteración no debe agregar demasiada funcionalidad para justificar el lanzamiento del producto al mercado, pero la meta es tener una versión de prueba al final de cada iteración. Cada vez que se finaliza una iteración el proyecto debe pasar a manos de los revisores, que en este caso puede ser cualquier persona ajena al desarrollo del proyecto y lo evalúa, comentando las cosas que más le ha llamado la atención y aconsejando cambios en cualquiera de las funcionalidades que se han implantado en esta nueva iteración[3].

Se puede observar como el proyecto comenzó sólo con el mapa del editor y a medida de que fueron avanzando las iteraciones o incrementos se fueron añadiendo nuevas funcionalidades y pruebas, hasta conseguir un producto final, siempre teniendo en cuenta las opiniones de las personas que probaban el proyecto, que en mi caso, podía ser cualquier persona de la familia que conociera el proyecto.

4.2. Análisis del sistema

El objetivo del análisis de sistemas es realizar un **Documento de Especificación de Requisitos**, que especifique los requisitos qué tiene que cumplir el sistema.

Durante esta fase se han tenido en cuenta algunos requisitos que consideramos fundamentales, con el fin de incrementar la probabilidad de éxito de implantación de nuestra aplicación. Podemos estructurar

dichos requisitos de la siguiente manera:

- **Calidad**. La permisividad mostrada por los usuarios ante los errores en aplicaciones es limitada; enlaces erróneos o información des-actualizada provocan la pérdida de usuarios de la aplicación. Es por ello que en el desarrollo de este tipo de aplicaciones es primordial disponer de mecanismos exhaustivos de control de calidad que minimicen las posibilidades de fracaso de la aplicación.
- **Velocidad**. El uso intensivo y continuo de las funciones de pintado sobre el editor provoca que la elección de la librería de desarrollo Qt de una velocidad adecuada, siendo una parte clave de diseño de dichas aplicaciones.
- **Eficiencia**. La aplicación debe hacer un buen uso de los recursos que manipula.
- **Importancia de la Interfaz**. La necesidad de implementar interfaces de usuario más intuitivas, capaces de capturar la atención del usuario y facilitar el acceso a la información a aquellos que poseen una habilidad limitada en el uso de aplicaciones informáticas.
- **Reutilización**. Define la capacidad de la biblioteca para ser reutilizada, en su totalidad o en parte, en nuevas aplicaciones.

4.3. Diagrama de Casos de uso

Siguiendo el proceso de desarrollo de software, se proponen los casos de uso necesarios para capturar los requisitos del sistema. Ésta es una técnica que fuerza a definir quienes son los actores (usuarios) de la aplicación y ofrece una manera intuitiva de representar la funcionalidad y aplicación en cada uno de los actores.
A continuación mostraremos los casos de uso más importantes junto con sus descripciones.

4.3.1. Caso de uso: Crear nuevo mapa normal

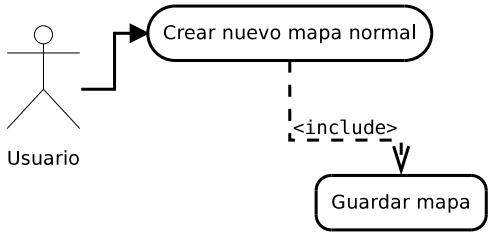

Figura 4.1: Diagrama de Caso de uso “Crear nuevo mapa normal”

Descripción del Caso de uso: *Crear nuevo mapa normal*

Descripción: Crea un nuevo mapa normal.
Actores: Usuario-Actor principal.
Precondiciones: Ninguna.

Escenario principal

1. El *Usuario* inicia la creación de un nuevo mapa normal.

2. El *Sistema* carga el diálogo de creación de un nuevo mapa normal y lo muestra en pantalla.

3. El *Usuario* introduce la anchura y la altura del patrón de tiles, y el número de tiles a lo largo y ancho.

4. El *Sistema* crea el nuevo mapa con las medidas indicadas por el usuario.

Escenario alternativo

1a. Si existe un mapa ya creado, que se encuentra en edición y se ha producido algún cambio desde el último guardado, se muestra un pequeño diálogo que permite guardar el mapa actual antes de crear el nuevo.

1. El *Usuario* decide guardar el mapa. Include *Guardar mapa*.

4.3.2. Caso de uso: Crear un nuevo mapa aleatorio

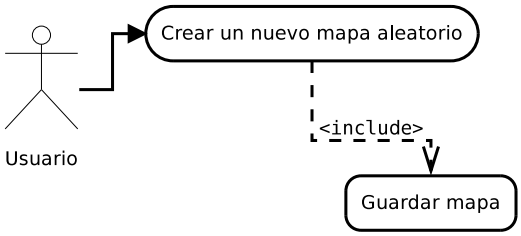

Figura 4.2: Diagrama de Caso de uso “Crear un nuevo mapa aleatorio”

Descripción del Caso de uso: *Crear un nuevo mapa aleatorio*

Descripción: Se crea un nuevo mapa aleatorio.
Actores: Usuario-Actor principal.
Precondiciones: Ninguna.

Escenario principal

1. El *Usuario* inicia la creación de un nuevo mapa aleatorio.
2. El *Sistema* muestra el diálogo correspondiente.
3. El *Usuario* introduce las medidas del mapa, selecciona el algoritmo y los tiles que compondrán el mapa.
4. El *Sistema* comprueba los datos, y crea el nuevo mapa aleatorio.

Escenario alternativo

2a. Si existe un mapa ya creado, que se encuentra en edición y se ha producido algún cambio desde el último guardado, se muestra un pequeño diálogo que permite guardar el mapa actual antes de crear el nuevo.

1. El *Usuario* decide guardar el mapa. Include *Guardar mapa*.

4.3.3. Caso de uso: Cargar mapa

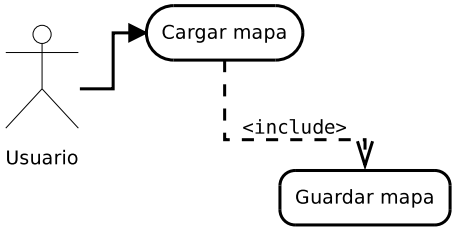

Figura 4.3: Diagrama de Caso de uso “Cargar mapa”

Descripción del Caso de uso: *Cargar mapa*

Descripción: Carga un mapa en el editor.
Actores: Usuario-Actor principal.
Precondiciones: Ninguna.

Escenario principal

1. El *Usuario* inicia el cargado de un mapa.
2. El *Sistema* muestra el diálogo de cargado.
3. El *Usuario* elige el archivo donde se encuentra guardado el mapa.
4. El *Sistema* carga el mapa y lo actualiza.

Escenario alternativo

2a. Existe un mapa en edición y se ha producido una modificación desde el último guardado.

1. Include *Guardar mapa*.

4a. El fichero escogido no es correcto.

1. El *Sistema* cancela la operación.

4.3.4. Caso de uso: Guardar mapa

Descripción del Caso de uso: *Guardar mapa*

Descripción: Guarda los datos del mapa.
Actor: Usuario-Actor principal.
Precondiciones: Existe un mapa en edición actualmente.

Escenario principal

1. El *Usuario* inicia el guardado de su mapa.
2. El *Sistema* muestra el diálogo de guardado del mapa.
3. El *Jugador* introduce el nombre con el que desea que se guarde el mapa.
4. El *Sistema* guarda los datos del mapa en un fichero con el nombre indicado.

Escenario alternativo

2a. El mapa había sido guardado previamente, y no se han producido cambios.

1. El *Sistema* no realiza el guardado.

2b. El mapa había sido guardado previamente, y se han producido cambios.

1. El *Sistema* guarda los datos del mapa en un fichero con el nombre con el que se guardo previamente.

4.3.5. Caso de uso: Rellenar mapa

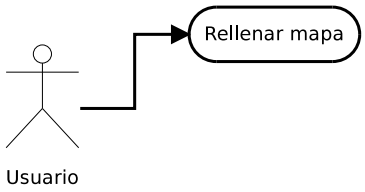

Figura 4.4: Diagrama de Caso de uso “Rellenar mapa”

Descripción del Caso de uso: *Rellenar mapa*

Descripción: El Usuario elige rellenar una posición del mapa.
Actor: Usuario-Actor principal.
Precondiciones: La opción de relleno está activada y existe un tile seleccionado.

Escenario principal

1. El *Usuario* elige la posición donde desea rellenar el mapa.
2. El *Sistema* guarda el número que corresponde con el tile seleccionado y luego actualiza el mapa.

Escenario alternativo

2a. La posición escogida se encuentra fuera de los límites del mapa.

1. El *Sistema* no lleva a cabo el relleno.

2b. En la posición escogida existe ya un tile.

1. El *Sistema* no lleva a cabo el relleno.

4.3.6. Caso de uso: Borrar mapa

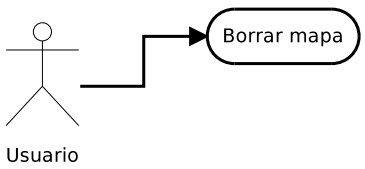

Figura 4.5: Diagrama de Caso de uso “Borrar mapa”

Descripción del Caso de uso: *Borrar mapa*

Descripción: Borrar un tile del mapa en edición.
Actores: Usuario-Actor principal.
Precondiciones: La opción de borrado esta activada.

Escenario principal

1. El *Usuario* selecciona la posición donde va a borrar el tile.

2. El *Sistema* comprueba la posición, borra el tile y luego actualiza el mapa.

Escenario alternativo

2a. En la posición no existe un tile.

    1. El *Sistema* cancela la operación.

2b. La posición escogida no se encuentra dentro de los límites del mapa.

    1. El *Sistema* cancela la operación.

4.3.7. Caso de uso: Mover tile

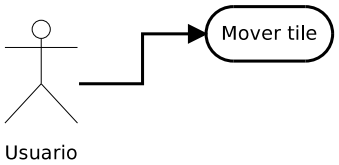

Figura 4.6: Diagrama de Caso de uso “Mover tile”

Descripción del Caso de uso: *Mover tile*

Descripción: Mover un tile de posición en el mapa en edición.
Actores: Usuario-Actor principal.
Precondiciones: Ninguna.

Escenario principal

1. El *Usuario* selecciona la posición donde se encuentra el tile.

2. El *Sistema* comprueba que la posición es correcta.

3. El *Usuario* elige la nueva posición.

4. El *Sistema* comprueba la nueva posición,realiza el movimiento y actualiza el mapa.

Escenario alternativo

2a. En la posición no existe un tile.

1. El *Sistema* cancela la operación.

2b. La posición escogida no se encuentra dentro de los límites del mapa.

1. El *Sistema* cancela la operación.

4a. En la posición escogida existe un tile.

1. El *Sistema* cancela la operación.

4b. La posición escogida no se encuentra dentro de los límites del mapa.

1. El *Sistema* cancela la operación.

4.3.8. Caso de uso: Importar un tile a la lista de tiles

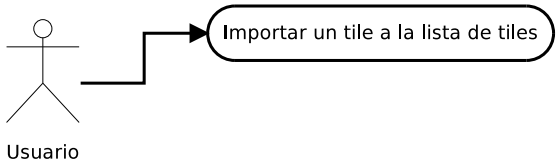

Figura 4.7: Diagrama de Caso de uso "Importar un tile a la lista de tiles"

Descripción del Caso de uso: *Importar un tile a la lista de tiles*

Descripción: Se importa un tile a la lista tiles.
Actores: Usuario-Actor principal.
Precondiciones: Existe un mapa en edición.

Escenario principal

1. El *Usuario* inicia la importación de un tile.
2. El *Sistema* abre el diálogo correspondiente.
3. El *Usuario* elige el fichero que desea importar.
4. El *Sistema* comprueba el fichero, inserta el tile en la lista y la actualiza.

Escenario alternativo

4a. El fichero no es correcto.

1. El sistema cancela la operación.

4.3.9. Caso de uso: Eliminar un tile de la lista tiles

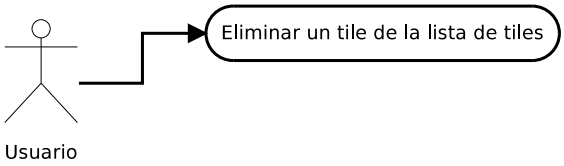

Figura 4.8: Diagrama de Caso de uso “Eliminar un tile de la lista tiles”

Descripción del Caso de uso: *Eliminar un tile de la lista tiles*

Descripción: Se elimina un tile de la lista de tiles.
Actores: Usuario-Actor principal.
Precondiciones: Existe tiles en la lista de tiles.

Escenario principal

1. El *Usuario* selecciona el tile que desea y lo elimina.
2. El *Sistema* actualiza el estado de la lista de tiles.

Escenario alternativo

4a. El fichero no es correcto.

1. El sistema cancela la operación.

4.3.10. Caso de uso: Cargar tileset

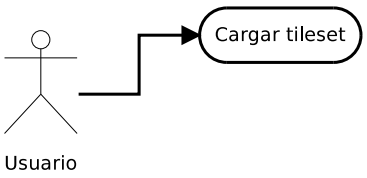

Figura 4.9: Diagrama de Caso de uso “Cargar tileset”

Descripción del Caso de uso: *Cargar tileset*

Descripción: Se carga un tileset en la lista tiles.
Actores: Usuario-Actor principal.
Precondiciones: Existe un mapa en edición.

Escenario principal

1. El *Usuario* inicia el cargado de un tileset.

2. El *Sistema* abre el diálogo correspondiente.

3. El *Usuario* elige el fichero que desea cargar y introduce la separación entre cada tile que compone el tileset.

4. El *Sistema* comprueba el fichero y la separación, e inserta los tiles del tileset en la lista tiles.

Escenario alternativo

4a. El fichero no es correcto.

1. El sistema cancela la operación.

4b. La separación no es correcta.

1. El sistema cancela la operación.

4.3.11. Caso de uso: Deshacer

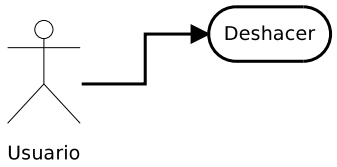

Figura 4.10: Diagrama de Caso de uso “Deshacer”

Descripción del Caso de uso: *Deshacer*

Descripción: Deshacemos la última acción realizada sobre el mapa.
Actores: Usuario-Actor principal.
Precondiciones: Se ha realizado alguna acción.

Escenario principal

1. El *Usuario* inicia la herramienta deshacer.

2. El *Sistema* recupera el estado anterior del actual y actualiza el mapa.

Escenario alternativo

2a. No hay mas acciones para deshacer.

1. El sistema no lleva a cabo la operación.

4.3.12. Caso de uso: Rehacer

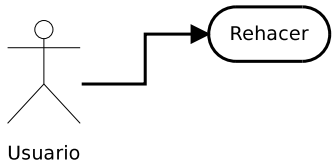

Figura 4.11: Diagrama de Caso de uso “Rehacer”

Descripción del Caso de uso: *Rehacer*

Descripción: Rehacemos la última acción deshecha sobre el mapa.
Actores: Usuario-Actor principal.
Precondiciones: Se ha realizado la acción deshacer.

Escenario principal

1. El *Usuario* inicia la herramienta rehacer.
2. El *Sistema* recupera el estado que anteriormente fue deshecho y actualiza el mapa.

Escenario alternativo

2a. No hay más acciones que rehacer.

1. El sistema no lleva a cabo la operación.

4.3.13. Caso de uso: Modificar zoom

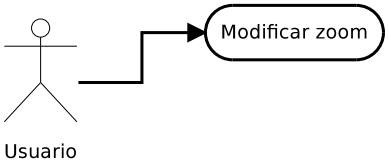

Figura 4.12: Diagrama de Caso de uso “Modificar zoom”

Descripción del Caso de uso: *Modificar zoom*

Descripción: Se modifica el zoom del mapa.
Actores: Usuario-Actor principal.
Precondiciones: Existe un mapa en edición.

Escenario principal

1. El *Usuario* modifica el zoom, aumentándolo o reduciéndolo.

2. El *Sistema* aplica la modificación, y actualiza el mapa.

Escenario alternativo

2a. No se puede ampliar mas.

1. El sistema cancela la operación.

2b. No se puede reducir mas.

1. El sistema cancela la operación.

4.3.14. Caso de uso: Seleccionar

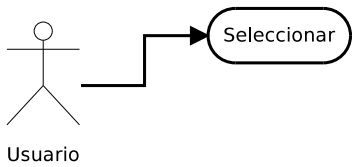

Figura 4.13: Diagrama de Caso de uso “Seleccionar”

Descripción del Caso de uso: *Seleccionar*

Descripción: Se selecciona una región del mapa.
Actores: Usuario-Actor principal.
Precondiciones: Existe un mapa en edición.

Escenario principal

1. El *Usuario* selecciona la región del mapa que desea.

2. El *Sistema* comprueba la selección escogida.

Escenario alternativo

2a. La posición de la región no es correcta.

1. El sistema cancela la operación.

4.3.15. Caso de uso: Copiar

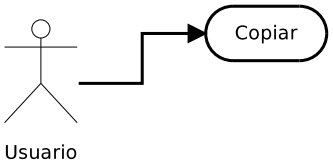

Figura 4.14: Diagrama de Caso de uso “Copiar”

Descripción del Caso de uso: *Copiar*

Descripción: Se copia una región del mapa.
Actores: Usuario-Actor principal.
Precondiciones: Existe un mapa en edición y una región seleccionada.

Escenario principal

1. El *Usuario* activa el copiado de la selección.

2. El *Sistema* copia el contenido de la región.

4.3.16. Caso de uso: Cortar

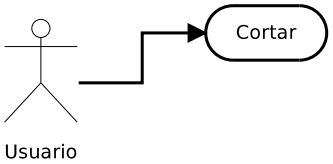

Figura 4.15: Diagrama de Caso de uso “Cortar”

Descripción del Caso de uso: *Cortar*

Descripción: Se copia y luego elimina una región del mapa.
Actores: Usuario-Actor principal.
Precondiciones: Existe un mapa en edición y una región seleccionada.

Escenario principal

1. El *Usuario* activa el cortado de la selección.

2. El *Sistema* copia el contenido de la región y luego la elimina.

4.3.17. Caso de uso: Pegar

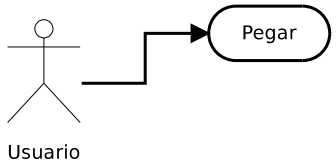

Figura 4.16: Diagrama de Caso de uso “Pegar”

Descripción del Caso de uso: *Pegar*

Descripción: Se pega una región del mapa.
Actores: Usuario-Actor principal.
Precondiciones: Existe un mapa en edición y una región copiada.

Escenario principal

1. El *Usuario* activa el pegado de la selección en la posición que desee.
2. El *Sistema* pega el contenido en la posición indicada.

Escenario alternativo

2a. La posición escogida no es correcta.

1. El sistema cancela la operación.

4.3.18. Caso de uso: Redimensionar mapa

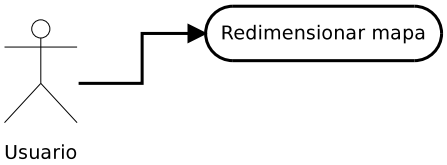

Figura 4.17: Diagrama de Caso de uso “Redimensionar mapa”

Descripción del Caso de uso: *Redimensionar mapa*

Descripción: Se redimensiona el mapa actual con nuevas medidas.
Actores: Usuario-Actor principal.
Precondiciones: Existe un mapa en edición.

Escenario principal

1. El *Usuario* inicia la redimensión del mapa.

2. El *Sistema* muestra el diálogo de redimensión.
3. El *Usuario* introduce las nuevas medidas del mapa.
4. El *Sistema* comprueba las medidas introducidas, redimensiona el mapa y lo actualiza.

Escenario alternativo

4a. Las medidas son incorrectas.

1. El sistema cancela la operación.

4.4. Modelo estático del sistema

El diseño conceptual se base en el análisis de requisitos del pase anterior.

El modelo conceptual propone construir un modelo de clases con estos objetos, ignorando los aspectos de navegación, presentación e interacción. Los principales elementos de modelado son; las clases, asociaciones y paquetes.

En nuestro caso, vamos a seguir un proceso de modelado orientado a objetos, basado en UML. Los pasos que seguiremos son:

- Distinguir las clases, como VentanaPrincipal o Mapa.
- Especificar los atributos más importantes y funcionamiento.
- Determinar las asociaciones entre las clases.
- Agregar las clases e identificar la composición de estas.
- Definir las restricciones de los métodos.

Diagrama de clases

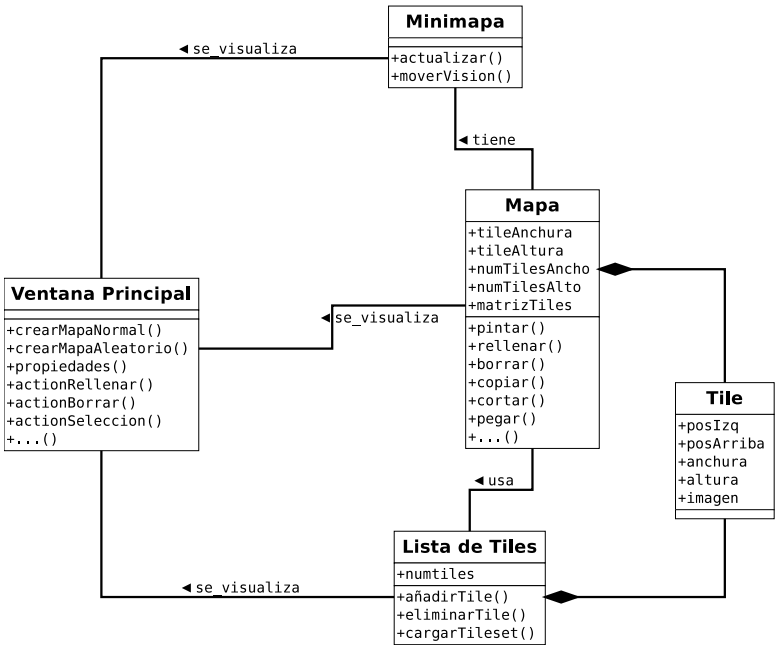

Figura 4.18: Diagrama de Clases

En el diagrama anterior vemos que la clase central es **Mapa** ya que es en la que realiza las funciones de edición del mapa. Otro papel muy importante lo recibe la clase **Ventana Principal** ya que es la que se encarga de controlar la interacción del usuario con la selección de las herramientas que ofrece el editor, aparte de las funciones de visualización de cada una de las partes de éste.

La clase **Tile** se muestra para que el lector pueda ver con mayor claridad como se estructura el editor.

4.5. Funciones del producto

ETiles está formado por un conjunto de herramientas que se aplican sobre la base del mapa a través de la interfaz que ofrece. Por ello primero veremos las funciones básicas que se puede realizar sobre el mapa y luego detallaremos herramienta a herramienta.

4.5.1. Interfaz gráfica

Como en la mayoría de programas la interfaz gráfica es una de las partes más importantes, y en éste no es una excepción. ETiles ofrece una interfaz sencilla y visiblemente agradable para que el usuario se encuentre a gusto en todo momento mientras edita su mapa. Aunque el usuario no haya utilizado nunca ningún editor del mismo estilo no tendrá ningún tipo de problema a la hora de usarlo ya que es totalmente intuitivo. Además en caso de que no se encuentre a gusto, puede personalizar las ventanas de visión moviéndolas a la posición que desee.

Ya que podremos gestionar toda la aplicación, es importante que el usuario se sienta cómodo con el entorno y aprenda fácilmente a manejar todas sus funcionalidades, ya que está orientada a cualquier tipo de persona. Por ello el usuario encontrará accesos rápidos a todas las herramientas del editor.

4.5.2. Ventana de edición del Mapa

Nos encontramos en la parte más importante del editor. En la ventana de edición del mapa es el lugar donde el usuario final visualiza uno a uno todos los tiles que forman su mapa y donde se aprecian todos los cambios que van sucediendo. Como ésta es la zona donde el usuario pasará el mayor tiempo durante el desarrollo de su mapa hay que hacer hincapié en la idea de hacerla lo más agradable posible. Por ello se provee de una serie de funciones para mejorar la calidad en el tratado de los tiles:

- La rejilla, opción que se utiliza para separar cada tile vertical y horizontalmente de sus vecinos ayudándonos a definir el tamaño de cada tile y su posición.
- En la zona inferior izquierda del mapa podremos observar que se nos muestra las coordenadas del tile sobre el que se encuentra el cursor a medida que movemos el ratón, función que puede apreciar el usuario cuando se trata de mapas grandes y se tiene el mapa muy estructurado.
- Cuando el mapa que nos ocupa exceda las dimensiones de la ventana de edición, aparecerá el scroll correspondiente permitiéndonos desplazarnos por todo el mapa mediante las barras de desplazamiento.

Además de las funciones de visualización, la ventana de edición permite realizar todas las acciones de edición disponibles en el editor. Si no tenemos ninguna acción seleccionada podemos realizar sólo dos de ellas:

- En caso de querer pintar un tile en la posición que indica el cursor, debemos de hacer click con el ratón, siempre y cuando, no exista ya un tile en dicha posición, y se tenga un tile seleccionado en la lista.
- En caso de querer mover un tile existente sólo debemos de seleccionarlo y arrastrarlo hacía la nueva posición, la cual debe estar vacía.

4.5.3. Lista de tiles

La lista de tiles es la encargada de almacenar los tiles que se importan al editor para crear nuestros mapas. Estos tiles se visualizarán en forma de pequeños iconos escalados a un tamaño prefijado dentro de la lista. Entre la funciones que ofrece se encuentran:

- Añadir o importar tile. Sólo necesitamos una imagen e indicar la ruta donde se encuentra para que la operación se lleve a cabo.
- Eliminar tile. Para poder eliminar un tile de la lista, primero debemos seleccionarlo y luego hacer click sobre el botón.
- Cargar Tileset. Cuando queramos cargar varios tiles de una sola vez hay que utilizar esta opción. Para ello, hay que indicar la ruta, y el borde de separación entre cada tile.

4.5.4. Barra de herramientas

La Barra de herramientas de acceso rápido ubicada en el rincón superior izquierdo de ETiles es una herramienta que nos permite obtener acceso rápido a las funciones más frecuentemente usadas del programa.

Los iconos usados hacen más fácil saber que es lo que hace cada acción. Se ha procurado utilizar botones similares para los mismos comandos usados en la mayoría de editores, todo para que el usuario se encuentre familiarizado desde un principio con el editor.

4.5.5. Herramientas guardar y cargar

Una vez que hemos hablado de las funciones básicas que realiza nuestro editor de mapas nos centraremos en cada una de las herramientas que nos aporta ETiles.

Como todo editor, ETiles ofrece poder guardar sus mapas editados. A la hora de guardar tenemos dos opciones:

- Guardar como: Esta opción se utiliza o bien cuando es la primera vez que realizamos un guardado o cuando queremos guardar el mapa especificando nombre y formato.
- Guardar: Esta opción se utiliza una vez que ya hemos realizado al menos un “Guardar como” si el mapa no es nuevo o es un mapa cargado previamente, almacenándose los cambios realizados de forma automática. Si no, llamará a la función “Guardar como”. Mencionar que esta operación sólo se llevará a cabo siempre y cuando se haya hecho algún cambio sobre mapa, ya que si no estaríamos sobreescribiendo el archivo de texto con el mismo contenido desperdiciando recursos en ello.

Y dependiendo del formato que elijamos:

- En caso de que elijamos el formato por defecto de ETiles, que llevará la extensión MAP, la matriz de datos con el número referente al tile que ocupa cada posición será guardado en un fichero.
- En caso de que elijamos otro de los existente formatos ya sea JPG, BMP, GIF, etc, unirá todos los tiles de forma que el mapa parezca una sola imagen.

Para la función cargar mapa de ETiles debemos haber guardado antes algún mapa con extensión MAP ya que es el único formato que permite su lectura.

4.5.6. Herramientas rellenar y borrar

Nos dedicamos ahora a otras dos herramientas que forman parte de la edición del editor.

La herramienta rellenar ofrece al usuario la facilidad de pintar su mapa de forma rápida. Con la función básica del editor debemos ir haciendo click en cada posición donde queremos colocar el tile seleccionado, lo que para mapas grandes puede ser una tarea ardua, costosa y puede llevar a colmar la paciencia del usuario desechando el intento de desarrollar su mapa. Por ello esta función mediante el movimiento del ratón, que es mucho más rápido, irá rellenando cada posición de forma automática.

La herramienta borrar funciona exactamente del mismo modo que rellenar excepto que eliminará el tile de la posición sobre la que se encuentra el cursor, siempre y cuando, no se encuentre vacía.

4.5.7. Herramientas selección, copiar, cortar y pegar

Nuestro editor ofrece la función selección, esta herramienta ofrece enormes posibilidades de manipulación de los tiles puesto que el área seleccionada se puede modificar sin afectar a otras. La selección es de forma rectangular, por tanto, para seleccionar un rectángulo se mantiene pulsado el botón izquierdo del ratón mientras se arrastra. Al soltar el botón la región quedará seleccionada, efecto que el usuario podrá apreciar en pantalla.

Gracias a la selección, que aisla una zona precisa del mapa, podremos realizar el resto de operaciones, ya sea copiar, cortar, pegar o eliminar.

4.5.8. Herramientas zoom, deshacer y rehacer

Otra de las funciones que no puede faltar en cualquier editor son las relacionadas con el zoom, con la que podemos ampliar y reducir la escala en la que se muestra la imagen. Muchas veces tendremos que realizar esta acción, ampliando el zoom para obtener mayor detalle de los píxeles de la imagen y reduciendo el zoom para obtener una visión del gráfico más general. A diferencia que ocurre con una imagen única, aquí hay que escalar tile a tile, aunque el resultado sigue siendo el mismo.

Como hemos comentado ya, ETiles ofrece las herramientas deshacer y rehacer, para ello el editor va almacenando cada una de las operaciones hechas por el usuario hasta un número configurable. Si el usuario se arrepiente de algún cambio, por muy anterior que sea, el editor le permite revertir todos los cambios hechos hasta el número configurado. Rehacer es por consiguiente, revertir algo revertido. De esta manera en el momento que se haga algún tipo de acción y no interese o no guste como ha quedado, se ahorrará mucho trabajo.

4.5.9. Herramienta de redimensión y propiedades

La función de redimensión permite al usuario redimensionar el mapa que esté editando en ETiles. El usuario podrá agrandar o disminuir las medidas a su antojo asumiendo la responsabilidad de su uso, ya que si decidimos disminuir el tamaño del mapa, el usuario tiene que tener en cuenta que los tiles que se encuentran fuera de los límites del nuevo mapa se borrarán permanentemente, debido a que no se puede recuperar.

De nuevo sobresale el aspecto de la personalización del editor gracias a la herramientas propiedades, uno de los principios más importantes de la aplicación, y que sin duda será agradecido por los usuarios que utilicen el sistema.

4.5.10. Mapas aleatorios

Normalmente cuando creamos un nuevo mapa el trabajo de desarrollo en la edición suele ser un trabajo pesado, que requiere de imaginación y de mucho esfuerzo. Crear uno de éstos mapas desde cero es una ardua tarea, así que un buen generador de mapas puede ser un muy buen compañero. Ésta es la idea básica de la función que se ofrece en el editor, permitir generar nuevos mapas de forma fácil y rápida, manteniendo una mínima calidad y coherencia en el resultado final.

Para crearlos, el usuario deberá indicar las medidas del mapa, los tiles que componen el mapa y el algoritmo que desea usar para crear la matriz de datos, además de si prefiere continentes frente a islas, uniforme frente a variado o agua frente a tierra.

Esta herramienta proporciona la posibilidad de que cualquier persona pueda agregar nuevos algoritmos de creación de mapas aleatorios que mejoren los ya existentes con el fin de lograr una herramienta más perfeccionada.

4.5.11. Minimapa

Una de las mayores comodidades de nuestro editor es el minimapa. Nos ofrece una imagen completa del mapa que estamos editando pero con un tamaño escalado, esto quiere decir que no se apreciará detalle en la imagen pero puede servir de mucha ayuda ya que nos permite orientarnos a través del mapa. Esto quiere decir que en cualquier momento podemos acceder a cualquier zona del mapa en la ventana de edición haciendo click aproximadamente sobre la posición escalada en el minimapa.

El minimapa se actualiza sólo en la zona que ha sido actualizada en el mapa de edición, es decir, se actualiza cuando hacemos alguna una modificación.

Capítulo 5

Diseño

La fase de Diseño se puede definir como una actividad consistente en aplicar distintas técnicas y principios con la finalidad de definir un sistema con suficiente detalle para que se pueda implementar. El diseño debe proporcionar una completa idea de lo que es el Software, enfocando los dominios de datos, funcional y comportamiento desde el punto de vista de la implementación.

Por tanto después de realizar la fase de análisis con todos los requisitos bien definidos, la tarea de diseño es simple. Sólo tenemos que definir los diferentes módulos que componen el modelo, junto con sus operaciones más importantes y una vez hecho esto pasar a la fase de implementación.

Esta aplicación ha sido diseñada siguiendo el patrón Modelo Vista Controlador (MVC), el cual es un patrón de arquitectura software que separa los datos de una aplicación(modelo), la interfaz de usuario(vista), y la lógica de control(controlador) en tres componentes distintos. En nuestro caso, la **vista** la forman las interfaces gráficas de cada clase. El **modelo** es el conjunto de estructuras dentro de las clases que almacenan la información, y el **controlador** es el responsable de gestionar los eventos de entrada de la vista.

Todo esto quiere decir que el diseño de la aplicación dividirá la interfaz gráfica del modelo de datos propuesto en la anterior etapa. Ambos se mantendrán conectados a través de los módulos controladores correspondientes.

A continuación detallaremos cada uno de ellos.

5.1. Ventana principal

La ventana principal será la encargada de proporcionar el marco principal del editor. En ella se inicializarán todos los componentes gráficos necesarios para que la aplicación funcione, gestionará las peticiones del usuario, comunicará los resultados al usuario y permitirá el tratado de las diferentes ventanas, diálogos, menús y botones.

Para conseguir todo esto, la ventana principal proporciona una serie de funciones capaces de mostrar la vista del editor.

- **VentanaPrincipal::VentanaPrincipal()**. Constructor de la clase ventana principal en el que se crea la superficie de la pantalla donde se visualizará el editor. Además se inicializan menús y la barra de herramientas.

- **VentanaPrincipal::subVentanaMapa()**. Crea y devuelve la ventana donde se visualizará nuestro mapa de edición.
- **VentanaPrincipal::subVentanaMinimapa()**. Crea y devuelve la ventana donde se visualizará nuestro minimapa.
- **VentanaPrincipal::subVentanaTiles()**. Crea y devuelve la ventana donde se visualizará nuestra lista de tiles.

Estas funciones se enlazan con cada una de las interfaces propias de cada módulo.

Aparte de todo esto, la ventana principal tendrá la responsabilidad de crear las diferentes instancias de las partes editor que forman la aplicación completa, ya que es la que recibe, a través de los diferentes diálogos que ofrece, todos los datos necesarios para su creación.

A continuación se describen las funciones más importantes.

- **VentanaPrincipal::nuevoMapaNormal(anchura,altura,numlargo,numalto)**. Crea la instancia del mapa según el patrón de tile deseado.
- **VentanaPrincipal::nuevoMapaAleatorio(anchura,altura,numlargo,numalto,tiles,algoritmo,indicaciones)**. Crea la instancia del mapa según el patrón de tile, los tiles que lo forman, el algoritmo a aplicar, y las indicaciones sobre el tipo de mapa que se desea.

Por último, como hemos comentado anteriormente, la ventana principal es la responsable de tratar los eventos correspondientes a la activación de las distintas herramientas que componen el editor, y conectarlas con las funciones que realizan la tarea sobre el mapa. Para ello se necesitan las siguientes funciones controladoras.

- **VentanaPrincipal::actionNuevo()**. Llama a la función que se encarga de crear los mapas normales.
- **VentanaPrincipal::actionNuevoAleatorio()**. Llama a la función que se encarga de crear los mapas aleatorios.
- **VentanaPrincipal::actionGuardar()**. Llama a la función que se encarga de guardar el mapa actual automáticamente.
- **VentanaPrincipal::actionGuardarComo()**. Llama a la función que se encarga de guardar el mapa con el formato y el nombre que deseemos.
- **VentanaPrincipal::actionCargar()**. Llama a la función que se encarga de cargar el contenido de un mapa guardado en un fichero.
- **VentanaPrincipal::actionRellenar()**. Función que activa la herramienta de relleno sobre el mapa.
- **VentanaPrincipal::actionBorrar()**. Función que activa la herramienta de borrado sobre el mapa.
- **VentanaPrincipal::actionSeleccionar()**. Función que activa la herramienta de selección sobre el mapa.
- **VentanaPrincipal::actionPegar()**. Función que indica que se a producido una acción de pegado sobre el mapa.

- **VentanaPrincipal::actionCopiar()**. Función que indica que se a producido una acción de copiado sobre el rectángulo de selección del mapa.
- **VentanaPrincipal::actionCortar()**. Función que indica que se a producido una acción de cortado sobre el rectángulo de selección del mapa.
- **VentanaPrincipal::actionEliminar()**. Función que indica que se a producido una acción de eliminación sobre el rectángulo de selección del mapa.
- **VentanaPrincipal::actionRejilla()**. Función indica que el usuario a decidido pintar o borrar la rejilla del mapa.
- **VentanaPrincipal::actionZoomIn()**. Función que indica que el tamaño del mapa debe ampliarse.
- **VentanaPrincipal::actionZoomOut()**. Función que indica que el tamaño del mapa debe reducirse.
- **VentanaPrincipal::actionDeshacer()**. Función que activa la herramienta deshacer sobre el mapa.
- **VentanaPrincipal::actionRehacer()**. Función que activa la herramienta rehacer sobre el mapa.
- **VentanaPrincipal::actionRedimensionar()**. Función que abre el diálogo de redimensión y que realiza la redimensión del mapa.
- **VentanaPrincipal::actionPropiedades()**. Función que abre el diálogo de propiedades que permite cambiar algunas opciones que nos ofrece el editor.

5.2. Mapa

Este módulo estará formado por todos los datos necesarios para poder representar nuestro mapa.

Primero definimos las funciones necesarias que proporcionen la interfaz gráfica de nuestro mapa, es decir, la superficie donde se visualizará.

- **Mapa::Mapa(tanchura,taltura,numtilesancho,numtilesalto)**. Constructor de la clase mapa, se encarga de inicializar las matriz de datos que contendrá los tiles, para ello necesita recibir por parámetros las anchura y la altura del patrón de tiles, el número de tiles a lo ancho y el número de tiles a lo alto. Aparte inicializara cada variable interna con su valor por defecto.
- **Mapa::pintar(rectangulo)**. Esta función es la encargada de pintar todos elementos que existen en el mapa. Cada vez que se produzca una modificación con alguna de las herramientas que ofrece el editor, la función actualizará el rectángulo sobre el que se ha producido.

A continuación definimos todas las funciones que tienen la responsabilidad de modificar los datos de nuestro mapa, las cuales se conectan con las descritas anteriormente en el módulo controlador de la ventana principal.

- **Mapa::rellenar(fila,columna)**. Función que se encarga de modificar el contenido de la matriz en la posición indicada por los parámetros con el número de tile seleccionado en la lista de tiles.
- **Mapa::borrar(fila,columna)**. Función que se encarga de modificar el contenido de la matriz en la posición indicada por los parámetros al valor por defecto que indica una posición vacía.

- **Mapa::seleccionar(rectangulo)**. Función que selecciona el rectángulo indicado por el usuario a través del movimiento del ratón, almacenándolo para su posterior uso en otras herramientas.
- **Mapa::copiar(rectangulo)**. Función que encarga de copiar los tiles que se corresponden en el mapa con el rectángulo de selección, almacenándolos en una lista junto con su posición.
- **Mapa::cortar(rectangulo)**. Realiza la misma función que copiar, es decir, llamará a dicha función y luego eliminará el contenido con una llamada a la función eliminar.
- **Mapa::pegar(rectangulo,tiles)**. Esta función se encarga de pegar tile a tile el contenido del rectángulo de selección en la posición donde indica el cursor.
- **Mapa::eliminar()**. Función que elimina cada tile que forma parte del rectángulo de selección.
- **Mapa::rejilla()**. Función indica que el usuario a decidido pintar o borrar la rejilla del mapa.
- **Mapa::zoomIn()**. Función que amplía el tamaño del mapa, y por tanto, de cada tile individual.
- **Mapa::zoomOut()**. Función que reduce el tamaño del mapa, y por tanto, de cada tile individual.
- **Mapa::deshacer()**. Función que recupera el estado anterior del mapa.
- **Mapa::rehacer()**. Función que recupera el estado posterior del mapa, siempre y cuando se haya usado antes deshacer.
- **Mapa::redimensionar(tanchura,taltura,numtilesancho,numtilesalto)**. Función que modifica alguno o todos los parámetros que forman las medidas del mapa, actualizándose, y llamando a la función pintar para que se aprecie el resultado por pantalla.

5.3. Lista de tiles

La lista de tiles es la encargada de visualizar los tiles que utilizaremos para crear y desarrollar nuestro mapa.

Siguiendo el patrón de diseño, encontramos una serie de funciones que inicializará la lista y las diferentes herramientas dentro de ella, además de proprorcionar la vista de los tiles.

- **ListaTiles::ListaTiles()**. El constructor inicializa la interfaz que tendrá la lista, es decir, cómo se visualizarán los tiles, y la barra de herramientas propias, junto con sus acciones.
- **ListaTiles::actualizar()**. Dibuja la lista de tiles y actualiza el estado de ésta cada vez que se produzca una acción sobre ella.

A continuación se muestran las funciones controladoras principales que permiten gestionar las peticiones del usuario.

- **ListaTiles::actionAñadirTile()**. Función que activa el diálogo correspondiente para añadir un tile a la lista.
- **ListaTiles::actionEliminarTile()**. Función que activa la eliminación de un tile seleccionado.
- **ListaTiles::actionAñadirTileset()**. Función que activa el diálogo de carga de tilesets para añadirlo a la lista.

- **ListaTiles::actionSeleccionarTile()**. Función que activa la selección de un tile.

Cada una de estas funciones se conecta con su correspondiente función, que son las encargadas de realizar la modificando sobre el modelo de datos.

5.4. Minimapa

En este módulo se concentrarán las funciones necesarias para crear y mantener actualizado el minimapa.

Para proporcionar la vista del minimapa definimos las siguiente funciones.

- **Minimapa::Minimapa()**. Constructor que inicializa el tamaño del minimapa y la zona donde se dibujará.
- **Minimapa::actualizarZona(rectangulo)**. Función que se encarga de pintar la zona que se actualiza en el mapa sobre el minimapa.

Además necesitamos una función que permita controlar las acciones del usuario sobre el minimapa.

- **Minimapa::moverVision(posicion)**. Función que traslada la visión de la ventana del mapa según la posición que se haya pulsado sobre el minimapa.

5.5. Características de los usuarios finales

ETiles se ha diseñado de forma que los usuarios finales no necesiten ninguna noción en ningún campo concreto de la informática, evidentemente, se entiende que el usuario debe saber navegar por su sistema operativo y manejar los periféricos del PC (teclado y ratón) de forma básica.

La aplicación va dirigida en especial al público desarrollador de videojuegos, pero dado que la aplicación se ha realizado de forma que sea totalmente intuitiva y muy parecida gráficamente a la mayoría de editores, puede ser utilizada por cualquier persona que desee crear sus propios mapas para su uso personal. Esto no quita que un usuario experto pueda utilizar la herramienta ya que se ofrece todo lo necesario para que se puedan crear mapas de buena calidad.

Capítulo 6

Implementación

Durante la fase de implementación nos dedicaremos a codificar todo el sistema, analizado y diseñado en etapas anteriores, mediante un lenguaje de programación adecuado. En nuestro caso hemos elegido el lenguaje de programación C++, con la ayuda de la librería de desarrollo Qt.

C++ es un lenguaje de programación diseñado a mediados de los años 1980 por Bjarne Stroustrup. Según Wikipedia, la intención de su creación fue el extender al exitoso lenguaje de programación C con mecanismos que permitan la manipulación de objetos. En ese sentido, desde el punto de vista de los lenguajes orientados a objetos, el C++ es un lenguaje híbrido. Posteriormente se añadieron facilidades de programación genérica, que se sumó a los otros dos paradigmas que ya estaban admitidos (programación estructurada y la programación orientada a objetos). Por esto se suele decir que el C++ es un lenguaje multiparadigma. Una particularidad del C++ es la posibilidad de redefinir los operadores (sobrecarga de operadores), y de poder crear nuevos tipos que se comporten como tipos fundamentales. C++ permite trabajar tanto a alto como a bajo nivel siendo muy óptimo[4].

El motivo de la elección de C++, es que el grado de familiarización con este tipo de lenguaje es muy alto. Ya que en la Universidad de Cádiz es el lenguaje por el cual se rigen la mayoría de las asignaturas. Es un lenguaje muy eficiente y que nos ayuda con multitud de herramientas que nos serán útiles a lo largo de la fase de implementación.

Otra herramienta utilizada es la biblioteca Qt. Es una biblioteca multiplataforma para desarrollar interfaces gráficas de usuario. Es producido por la división de software Qt de Nokia. Qt está diseñado para usarse principalmente sobre el lenguaje C++, aunque ofrece herramientas para utilizarse sobre otros lenguajes. La biblioteca se distribuye, entre otras, bajo la licencia Lesser General Public License (LGPL), que es la que ha provocado el gran avance y evolución de la biblioteca. Además tiene un gran apoyo y uso, y se encuentra en continua actualización.

El motivo de la elección de esta biblioteca no es otro que su facilidad de aprendizaje y la extensa documentación que posee. Ofrece numerosas herramientas que permiten crear nuestros mapas con tamaños enormes y mantenga unas condiciones muy óptimas de eficiencia, Además, permite obtener una interfaz gráfica agradable y fácil de implementar[5].

La última herramienta que usamos es libSDL, Simple DirectMedia Layer (SDL) que es un conjunto de bibliotecas desarrolladas con el lenguaje C, normalmente usada en el desarrollo de videojuegos, que proporcionan funciones básicas para realizar operaciones de dibujado 2D, gestión de efectos de sonido y música, y carga y gestión de imágenes. La biblioteca también se distribuye bajo la licencia LGPL [6].

El motivo de desarrollar la biblioteca del editor para librería libSDL es que ya es conocida para mí, ya que al haber cursado la asignatura de Diseño de Videojuegos, cuyo profesor es uno de los tutores de este proyecto, es la herramienta que usamos para desarrollar el curso. Cómo principalmente el editor está orientado al público orientado al desarrollo de videojuegos, es interesante ofrecer distintas funciones que puedan utilizar los desarrolladores.

6.1. Desarrollo de ETiles

Para el desarrollo de ETiles, hemos usado *Qt Creator* es un Entorno de Desarrollo Integrado (IDE) que nos sirve para programar en C++ usando las librerías de Qt.

Entre las características que nos ofrece *Qt Creator*:

- Posee un avanzado editor de código C++.
- Soporta los lenguajes: C#/.NET Languages (Mono), Python: PyQt y PySide, Ada, Pascal, Perl, y Ruby.
- Posee también una Interfaz Gráfica de Usuario (GUI) integrada y diseñador de formularios.
- Herramienta para proyectos y administración.
- Ayuda sensible al contexto integrada.
- Depurador visual.
- Resaltado y auto-completado de código.

Qt Creator es distribuido bajo tres tipos de licencias: Qt Commercial Developer License, Qt GNU LGPL v. 2.1, Qt GNU GPL v. 3.0 y está disponible para las plataformas: Linux, Mac OSX; Windows, Windows CE, Symbian y Maemo.

6.2. Ventana Principal

En primer lugar hablamos de la clase que se encarga de proporcionar la interfaz gráfica a ETiles. Para ello la librería Qt ofrece una clase llamada **QMainWindow**, que provee un marco para gestión de la aplicación. La clase tiene su propio diseño pero que puede ser modificado a nuestro antojo, sólo debemos llamar a la funciones correspondientes. En este caso se inicializará con un tamaño 800x600 y se visualizará en el centro de la pantalla.

A continuación mostramos una imagen de cómo se estructuraría la ventana principal:

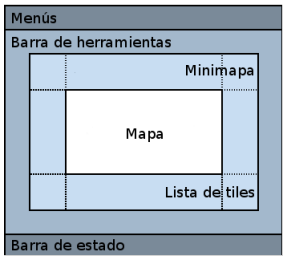

Figura 6.1: Distribución de la ventana principal

Como vemos en la zona central se ubicará nuestro mapa. Mientras no se cree un nuevo mapa, la ventana de visión aparecerá vacía. Al igual ocurre con las ventanas que almacenan la lista de tiles y el minimapa.

La ventana principal es la que permite al usuario crear los mapas, guardarlos, y cargarlos, además de implementar las funciones controladoras necesarias para acceder a cada una de las herramientas del editor. Ahora procederemos a explicar las funciones más importantes de esta clase.

La función `crearMapaNormal(medidas)`, es la que se encarga de inicializar la ventana de visión y creación de los mapas. Necesita una serie de parámetros del usuario por lo que se implementa un diálogo que permite introducir las siguientes medidas:

- La anchura del patrón del tile.
- La altura del patrón del tile.
- El número de tiles a lo ancho.
- El número de tiles a lo largo.

Una vez introducidas, la función crea una instancia de la clase **Mapa** con las dimensiones especificadas y crea un contenedor en el que se coloca como superficie central el mapa. Este contenedor no es más que una clase que proporciona el scroll necesario cuando las dimensiones sobrepasan los límites de la ventana de visión. Luego se inicializan la lista de tiles y el minimipa que le corresponden, relacionándose entre si.

La función `crearMapaAleatorio(parametros)`, es la encargada de crear los mapas aleatorios. Al igual que con la creación de los mapas normales, se diseña un diálogo que permita introducir cada uno de los parámetros que se citan a continuación, obviando los que ya hacen falta para crear un mapa normal:

- Los tiles que compondrán el mapa. Para insertarlos se proporciona una lista de tiles junto a su barra de herramientas, para que el usuario pueda observar de forma gráfica los tiles que va añadiendo.
- Porcentaje numérico para saber si prefiere un mapa con mayor cantidad de tierra frente a agua o viceversa.
- Porcentaje numérico para saber si prefiere un mapa con continentes o de islas.
- Porcentaje numérico para saber si prefiere un mapa uniforme frente a uno variado.

- Número entero que selecciona el algoritmo que desea utilizar para la creación de su mapa.

Introducidos todos los parámetros se devuelve la matriz con los tiles seleccionados en cada posición. Luego se inicializan todas las partes del editor y se copia el contenido de la matriz en la del mapa recién creado.

Puede que una de las funciones más complicadas de entender su funcionamiento sea ésta, y es que el proceso seguido puede resultar confuso. Por ejemplo, una de las técnicas usadas para la creación de estos mapas que proporciona el editor es la función **Perlin Noise**. Fundamentalmente se basa en funciones de tipo **Noise**, que no es más que un generador de números aleatorios. Se necesita un número entero como parámetro, y devuelve un número aleatorio basado en ese parámetro. Luego, se toman un conjunto de estas funciones, con diferentes frecuencias y amplitudes, combinándose entre ellas para crear un efecto **Perlin Noise**. En nuestro caso particular, haremos uso de la función para dos dimensiones, por tanto, a través de las coordenadas de cada tile que componen el mapa se genera un número pseudo-aleatorio que se corresponderá con un determinado tile de la lista de tiles. Además mediante las indicaciones del usuario se modifican la amplitud y la frecuencia para obtener el efecto deseado[7].

La función guardarProyecto(bool) es la encargada de guardar el mapa según el formato que se especifique. Para ello, se implementa un diálogo donde el usuario puede introducir el nombre con el que desea guardar el mapa y la extensión. Dependiendo de la extensión:

- Si la extensión es de tipo .map, que es la utilizada por el editor, vuelca en contenido de la matriz de tiles en un fichero, seguido de los tiles que la componen. Este formato permite almacenar las imágenes de los tiles dentro del fichero de manera comprimida de forma que siempre estén disponibles para el usuario y no dependan de un almacenamiento externo al programa.

- Dado que el formato anterior hace uso de las bibliotecas Qt, no es posible usar estos ficheros para cargar nuestros mapas desde otra biblioteca sin hacer uso de ésta. Por ello, se proporciona otro distinto con extensión .mbp, que vuelca el contenido de la matriz sobre un fichero de texto plano seguido de una línea por cada tile que existe en la lista tiles. Dicha línea está formada por la separación entre tiles, para el caso de un tileset o cero para tiles únicos, y la ruta del fichero de imagen que lo identifica.

- Si se elige otro tipo de formato para guardar el mapa en una nueva imagen, se pinta el mapa sobre una imagen de la clase **QPixmap** de la librería Qt, que nos permite con la función `save(filename)` guardar el mapa como una única imagen con el formato deseado.

El parámetro que recibe la función nos indica si se realizará la opción **Guardar como** o sólo **Guardar**.

La función `cargarProyecto()` es la encargada de cargar los mapas que han sido guardados previamente con la función `guardarProyecto(bool)`. El proceso seguido es el mismo que para guardar los mapas de extensión .map. Se lee la matriz de datos y los tiles del fichero, y luego se inicializan todas las partes del editor como si se tratara de un nuevo mapa.

Cabe destacar que todos los diálogos que se mencionan han sido creado con la herramienta *Qt Creator* de forma gráfica, y se accede a ellos mediante la clase donde se implementan.

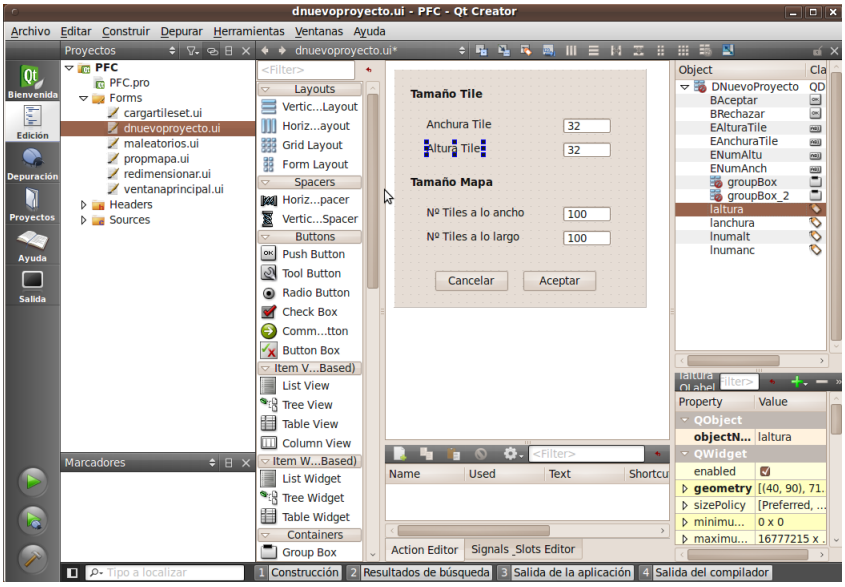

Figura 6.2: Ventana de edición de diálogos de Qt Creator

6.2.1. Barra de herramientas

Como observamos antes en la figura 6.1 la barra de herramientas se sitúa entre los menús y el resto de ventanas. Para crear la barra tenemos que ir añadiendo acción a acción, indicando el icono que lo representa, y su descripción. Luego es necesario conectar la acción de clickear sobre cada una de las herramientas con la función correspondiente que desempeña, pero teniendo en cuenta, que debemos comprobar que el mapa a sido ya creado para que no se produzcan errores.

6.2.2. Menús

Los menús se sitúan en la parte superior del editor. La implementación no conlleva ninguna dificultad ya que *Qt Creator* nos permite crearlos de forma gráfica. Sólo necesitamos indicar el nombre de cada una de las partes y su correspondiente tecla de acceso rápido. Una vez creado, debemos enlazar el evento de clickear sobre los menús con las funciones que les corresponden, comprobando al igual que en la barra de herramientas, que el mapa ya ha sido creado.

6.2.3. Barra de estado

La barra de estado se visualiza en la parte inferior del editor, esta barra solo posee una función, y es la de mostrar las coordenadas del tile sobre el que se encuentra el cursor. Para poder cambiar el contenido del texto de la barra de estado se crea una función que recibirá la cadena de texto y modificará el contenido cada vez que desplacemos el ratón.

6.3. Mapa

Esta es sin duda la clase más importante del editor, es el lugar donde se llevan a cabo todas las acciones que permite ETiles para editar nuestro mapa.

Para implementar la clase Mapa, se ha usado la clase **QLabel** que proporciona la librería Qt. Esta clase nos ofrece una superficie en la que poder pintar nuestro mapa de forma rápida y eficiente, además proporciona distintas funciones para el manejo del ratón que nos serán de enorme utilidad para nuestro desarrollo.

Antes de todo hay que tener en cuenta que el sistema de coordenadas que utiliza la biblioteca de desarrollo Qt es diferente al sistema cartesiano de coordenadas que normalmente solemos usar.

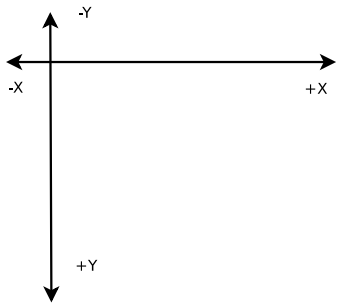

Figura 6.3: Sistema de coordenadas en Qt

Como se observa, el punto (0,0) se establece en la esquina superior izquierda, cualquier incremento positivo en el eje X supone un incremento hacia la derecha, mientras que en el eje Y producirá un incremento hacia abajo, a diferencia del sistema cartesiano que incrementaría hacia la derecha y arriba respectivamente.

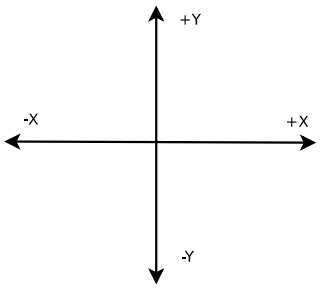

Figura 6.4: Sistema cartesiano

Por todo esto los índices para manejar la matriz de datos que almacena los tiles seguirán este sistema de coordenadas. Por ejemplo, si colocamos un tile en la posición (40,60), en un mapa de tiles de 20x20, se guardará en la posición (2,3).

Explicado el sistema de coordenadas comenzaremos con la creación del mapa, donde será necesario indicar el tamaño del patrón de tile (anchura y altura), y el número de ellos a lo ancho y largo. Las medidas finales se obtienen multiplicando ambos valores. Luego inicializaremos la matriz de enteros con valor negativo que corresponde con una posición vacía del mapa.

Una de las acciones que más quebraderos de cabeza a dado ha sido la forma de pintar nuestro mapa, y es que, la librería Qt ofrece multitud de superficies donde poder realizar esta acción pero no todas son igual de eficientes. La clase **QLabel** está preparada para mostrar imágenes de forma rápida y eficiente pero tiene algunos problemas. Entre ellos el más importante es que todos los efectos de pintado sólo se pueden realizar dentro de la función `paintEvent(rectangulo)`, por lo que cada vez que deseemos pintar un tile, un cuadro de selección, la rejilla, etc, deberemos llamar a esta función. Todo ésto conlleva a tener que diferenciar mediante variables distintos flujos dependiendo de la herramienta en la que nos encontremos, lo que hace que el código pueda resultar pesado y difícil de entender. Por tanto, vamos a explicar los pasos que se llevan a cabo:

1. Se comprueba si la rejilla está activa o no, en caso de que lo esté, al manejador encargado de pintar los tiles se le asignará un borde del color de la rejilla, ya que la rejilla no se pinta cada vez de forma completa, sino que es la unión de todos los bordes de los tiles que forman el mapa.

2. Pintamos la región a actualizar con el color de fondo del mapa, ya que cada vez que se llame a esta función el contenido del rectángulo es borrado automáticamente por la librería Qt. Como mínimo durante la ejecución se llegará hasta este paso, por ejemplo, cuando movemos el ratón y tenemos que actualizar la posición anterior del ratón para borrar el cuadro de selección.

3. Comprobamos si se trata de un movimiento del ratón, ya que es la acción que más se realizará por parte del usuario. Dentro del flujo existen tres posibles opciones, que se trate de un movimiento con la herramienta selección activada, por lo que se pintará el rectángulo de selección, de un simple movimiento, con lo cual se pintará el cuadro de resaltado sobre el tile que indica la posición del ratón, o de un movimiento con el pegado activado, pintándose en rectángulo de pegado.

4. Llegados a este punto sólo quedaría actualizar la región que indica el rectángulo, ya que la única opción que queda es que se haya producido un cambio en alguno de los tiles de nuestro mapa. Para ello comprobamos si el rectángulo se corresponde a más de un tile del mapa, en cuyo caso, debemos ir recorriendo con un bucle cada uno de ellos para pintarlos sobre el mapa.

Cabe mencionar que la llamada a la función `paintEvent(rectangulo)` será siempre controlada a través de otra función llamada `update(rectangulo)`. Esta última se utiliza cada vez que queremos realizar alguna acción de pintado sobre una región establecida por las dimensiones del rectángulo. Gracias a esta función se consigue una mayor eficiencia ya que controla el número de llamadas que se realizan evitando que se sobrecargue la función de pintado y como consecuencia se ralentice el editor.

Al igual que ocurre con la función de pintado, debemos sobrecargar las funciones relacionadas con los eventos que produce el ratón proporcionadas por la clase **QLabel**. Al existir sólo una única función por cada evento, nos encontramos con el mismo problema que antes, por lo que volvemos a tener que diferenciar entre distintos flujos dependiendo de la herramienta seleccionada en cada momento.

Para editar nuestro mapa sólo utilizaremos dos:

- La función `mousePressEvent(posicion)`: Cada vez que se produzca un evento click con el ratón se llamará a está función. Si está activada alguna de las acciones pegar, rellenar, borrar o seleccionar, se llamará a la función correspondiente. Si no se encuentra activada ninguna de ellas, y existe un tile en la posición, significa que vamos a desplazarlo. Para producir este efecto se hace uso de otra clase que proporciona la librería Qt, la clase **QDrag**, que nos ofrece un soporte para realizar la transferencia de datos, mostrándonos en todo momento el tile en movimiento

junto con el cursor del ratón. Por otro lado esta clase nos obliga a tener que definir un modelo de datos mediante la función `setData(id,datos)`, con la cual indicamos un nombre que nos identifique de forma única el tipo de datos que se aceptarán para las operaciones de la clase. Por tanto, cuando el usuario suelte el tile en la nueva posición, y se llame a la función `dropEvent(posicion,datos)`, tenemos que comprobar que el identificador se corresponde con el que hemos definido previamente, y luego realizar la operación correspondiente. Todo ésto se lleva a cabo para que no se produzcan errores cuando se realicen este tipo de acciones, ya que por ejemplo, si arrastramos cualquier objeto externo al programa dentro del mapa y soltamos el botón de ratón también será controlado por dicha función, por lo que debemos ignorar el evento.

- La función `mouseMoveEvent(posicion)`: Cada vez que se produzca un movimiento con el ratón se llamará a está función. Al igual que antes se diferenciará entre las herramientas rellenar y borrar, llamándose a sus respectivas funciones, y las herramientas pegar o seleccionar, las cuales actualizarán las posiciones antiguas y nuevas del cuadro de selección o pegado.

Una vez conocidos como se implementan los eventos del ratón, describiremos algunas de las funciones más complejas que implementan las herramientas del editor.

Quizás la que entraña mayor dificultad para su implementación sean las herramientas deshacer y rehacer. Como hemos comentado en anteriores capítulos, estas herramientas sólo tendrán efecto sobre las acciones de edición del mapa. La implementación se ha realizado a través de listas, por ello, ambas funciones tendrán asignada una primera lista de dos enteros, que designan el tipo de operación y el número de tiles que se han modificado en ella respectivamente. Primordialmente todas las operaciones que realiza el editor se basan en tres:

- La inserción de un tile, que se representará con el número 0.
- La eliminación de un tile, que se representará con el número 1.
- El movimiento de un tile a una nueva posición, que se representará con el número 2.

Cada vez que realicemos una operación que modifique de algún modo el mapa mediante alguna de estas operaciones tendremos que introducir ambos valores en la lista. Por ejemplo, si realizamos una opción de pegado de un rectángulo de cuatro tiles, insertamos en la lista deshacer el cero indicando que es una operación de inserción y cuatro indicando que son cuatro tiles. Además de esta lista, para la inserción y eliminación, harán falta otras dos, una primera donde se almacenan el número del tile que se está modificando, y una segunda donde se almacena la posición en la cual se ha producido la modificación. Por último si se trata de un movimiento, tendremos que almacenar la posición antigua y la nueva, en vez de el valor del tile, ya que dicha información permanecerá todavía en la matriz y no se perderá.

Una vez identificadas todas las listas necesarias, cada vez que llamemos a la función deshacer, dependiendo del tipo de operación que sea revertirá la acción accediendo a los valores de las listas. La función rehacer sigue la misma implementación sólo que los valores de las lista los obtiene desde la función deshacer.

Otra función interesante es la función `zoom(bool)`, que recibe un booleano que indica si se aumenta o disminuye el zoom. Para está función se dispone de una variable global(vector) que contiene todos los zoom posibles, por lo que solo debemos avanzar o retroceder a través del vector dependiendo de si el valor es verdadero o falso respectivamente. Una vez obtenido el nuevo valor del zoom, que puede ser del 20 %,50 %,75 %,100 %,125 %,150 %,200 %, se calculan los nuevos valores que tendrán las medidas del mapa, y de cada tile en particular. Éstas últimas se almacenarán en las variables `anchuraZ` y

`alturaZ`, que serán las que manejará el editor para las operaciones de edición, en vez de las medidas originales. Una vez calculado todo, se modifica el tamaño de la superficie del mapa y se llama a la función `update()`, para que pinte cada tile particular con las nuevas dimensiones.

Por último explicaremos otra función que posee cierta dificultad ya que no funciona exactamente del mismo modo que en otros programas de edición. Nos referimos a la función `copiar(rectangulo)`. A diferencia de otros programas, el copiado se realiza sobre un rectángulo, es decir, un conjunto de tiles que forma una zona rectangular. Como dentro de esta zona pueden existir o no los tiles, se utilizará una lista donde se almacenarán las coordenadas que ocupa cada tile a copiar dentro del rectángulo. Por ejemplo, si copiamos un rectángulo del mapa de 2x3 tiles, y sólo existen dos tiles en las posiciones (1,1) y (2,2), insertaríamos estos valores en la lista de coordenadas, aparte, en otra lista almacenamos el valor del tile. Por tanto, en la función `pegar()` se recorrerá cada elemento de la lista, calculando la posición de pegado a partir de la posición del cursor(donde se posicionará el rectángulo de pegado) y la lista de coordenadas. Siguiendo el mismo ejemplo, si el ratón se encuentra en las coordenadas (7,7) los dos tiles que se copiaron se pegarían en las posiciones (8,8) y (10,10), siempre y cuando, no exceda las dimensiones del mapa y no exista un tile ya en dichas coordenadas.

6.4. Lista tiles

La lista de tiles es la herramienta que nos permitirá almacenar y visualizar los tiles que importaremos para desarrollar nuestro mapa.

Siguiendo la etapa de diseño, primero se implementa una clase que provea el modelo de datos que seguirá la lista, utilizando para ello la clase **QAbstractListModel** que nos proporciona la librería Qt. Dentro de la clase declaramos una lista para almacenar las imágenes de los tiles y otra para la ruta del fichero donde se sitúan. Luego sobrecargamos las funciones de insertado, eliminación y demás de la clase.

A continuación usamos otra clase de la librería Qt, **QListView** para crear una clase que proporcionará la interfaz gráfica a la lista. En esta clase definiremos la forma en la que se mostrarán los tiles dentro de lista. Se ha decidido que se visualizarán en forma de iconos con tamaño 32x32 píxeles, con una separación de un píxel. Además, se implementa la barra de herramientas de la lista que ofrecerá realizar las tres acciones disponibles.

Para enlazar la interfaz con el modelo de datos, solo hay que hacer uso de la función `setModel(modelo)` de la clase que provee la interfaz.

6.5. Minimapa

Para la implementación del minimapa se crea una clase que hereda de la clase **QLabel** al igual que la clase **Mapa**. La razones de su uso son las mismas, la necesidad de visualizar una imagen de la forma más rápida y eficiente posible.

Como sabemos, el minimapa es una herramienta que aporta una mayor comodidad para el usuario, pero es opcional, ya que en cualquier momento el usuario puede dejar de visualizarlo.

La actualización del minimapa se realiza a través de dos funciones:

- La función `actualizar(datos)`, que recibe la posición que se ha modificado, el tile, y si se trata de una inserción o eliminación. Esta función calcula las dimensiones escaladas del rectángulo a pintar y llama a la función `pintar(rectangulo)`.
- La función `pintar(rectangulo)`, actualiza la región del mapa según la posición escalada que recibió de la función anterior, y pinta el tile si se trata de una inserción o lo borra en caso de una eliminación. Existe una tercera opción, y es que hay veces en que el minimapa se actualiza al completo, por ejemplo, cuando creamos un nuevo mapa aleatorio, por lo que es necesario pintar cada tile del minimapa. Dado que el minimipa no almacena ningún dato interno, sólo la imagen sobre la que se pinta, es necesario acceder a los datos de la matriz del mapa, por lo que produce que esta operación conlleve un alto coste.

6.6. Biblioteca libSDL

Para implementar la biblioteca se ha desarrollado una clase llamada **SDLEtiles** que proporciona las funciones necesarias para crear, cargar y pintar nuestro mapa bajo la librería libSDL. La representación interna es exactamente igual que la implementada para la clase **Mapa**, ya que usaremos una matriz de enteros para saber que tile ocupa cada posición.

La clase nos permitirá cargar los mapas que hemos exportados desde nuestro editor con el formato .mbp para poder usarlos en la biblioteca. A través del método `cargarMapa(fichero,rutaimagenes)` leemos los datos de la matriz y los almacenamos. Luego debemos cargar todas las imágenes de los tiles. Como sabemos, al exportar, cada línea de texto que sigue a los datos de la matriz se corresponde con un tile cargado. Esta línea se compone de un número que representa la separación entre tiles de un tileset, y un cero para un único tile. A cada número le sigue el nombre del fichero que contenía la imagen. Ya que sólo conocemos el nombre, es necesario que se pase por parámetro la ruta donde se encuentran todas las imágenes. Por tanto, para almacenarlos debemos diferenciar entre tileset y tile de la siguiente forma:

- Si se trata de un tile, cargamos la imagen con la función `IMG_Load(fichero)`, que nos devuelve una superficie del tipo `SDL_Surface*` donde se encuentra nuestra imagen cargada.
- Si se trata de un tileset, cargamos la imagen del tileset del mismo modo, luego a través de un bucle vamos descomponiendo la imagen en los tiles que la componen, creando una nueva superficie con la función `SDL_CreateRGBSurface()` de las dimensiones del tile, y copiando con la función `SDL_BlitSurface()` la zona del tileset que se corresponde con el tile.

Además de cargar los mapas, también podemos crear nuevos mapas aleatorios a través de su método `crearMapaAleatorio()` del mismo modo que ocurría en ETiles.

Para poder visualizar el mapa, se provee el método `pintarEnSuperficie(superficie)`, el cual recibe una superficie del tipo `SDL_Surface *` donde se realizará el pintado. La implementación se lleva a cabo con dos bucles anidados que van recorriendo la matriz, y usando la función `SDL_BlitSurface()` copiamos la imagen del tile en la posición que ocupa dentro de la superficie dada.

Capítulo 7

Pruebas

En el presente apartado nos dedicaremos a describir cada una de las pruebas que hemos ido realizando a cada una de las herramientas disponibles en ETiles, con el fin de comprobar que el sistema se comporta tal y como lo especificamos inicialmente, y por tanto, cumple todos los requisitos impuestos.

Cabe destacar que el sistema ha sido probado por terceras personas, con niveles básicos sobre informática, para asegurarme que la aplicación es sencilla de utilizar y para comprobar realmente que el sistema se comporta adecuadamente ante cualquier solicitud del usuario.

7.1. Pruebas realizadas

El proceso de prueba comienza con la generación de un plan de pruebas en base a la documentación del proyecto y a la documentación del software a probar. A partir de dicho plan, se diseñan los casos de prueba, se ejecutan y los resultados obtenidos se comparan con los resultados esperados. Una vez evaluados los resultados de las pruebas, pueden realizarse dos actividades:

1. Depurar los defectos

2. Analizar los errores

La depuración puede corregir o no los defectos. Si no consigue localizarlos, puede ser necesario realizar pruebas adicionales para obtener más información. Si se corrige un defecto, se debe volver a probar el software para comprobar que el problema está resuelto.

Dado que no se pueden probar todas las posibilidades de funcionamiento del software, la idea fundamental para el diseño de casos de prueba consiste en elegir aquellas posibilidades que, por sus características, se consideran representativas del resto. De esta forma se asume que, si no se detectan defectos en el software al ejecutar dichos casos, podemos tener cierto nivel de confianza (que dependerá de la elección de los casos) en que el programa no tiene defectos. La dificultad de esta idea está en saber elegir los casos que se deben ejecutar.

A continuación mostramos alguno de los casos de prueba más significativos de cada incremento.

7.2. Incremento 1: Requisitos Básicos del sistema

7.2.1. Caso de prueba: Inicialización de la aplicación

Descripción: En esta prueba comprobamos el funcionamiento de la aplicación al inicializarse, creándose la pantalla principal de ETiles.

Condiciones de ejecución: Ninguna.

Pasos seguidos:

1. Ejecutar la aplicación desde la terminal.
2. Maximizamos y minizamos la ventana.
3. Movemos la ventana a distintas posiciones.
4. Cerramos la aplicación.

Resultado esperado: La ventana se debe crear en la posición indicada y con las medidas establecidas, permitiendo las acciones de movimiento, maximización, minimización y cerrado sobre la ventana.

Evaluación de la prueba: Realizada y satisfactoria.

7.2.2. Caso de prueba: Funcionalidades de las ventanas de visión

Descripción: En esta prueba comprobamos que las ventanas que muestran el mapa, la lista de tiles y el minimapa se comportan de la forma esperada.

Condiciones de ejecución: Se ha ejecutado la aplicación.

Pasos seguidos:

1. Cerramos y abrimos cada una de las ventanas.
2. Movemos las ventanas a nuevas posiciones dentro de la aplicación.
3. Flotamos las ventanas como si se trataran de ventanas independientes.
4. Redimensionamos las ventanas.

Resultado esperado: La ventanas se puede mover a nuevas posiciones, redimensionar, flotar como ventanas independientes, cerrar y volver a abrir.

Evaluación de la prueba: Realizada y satisfactoria.

7.3. Incremento 2: Creación del mapa

7.3.1. Caso de prueba: Crear un nuevo mapa con medidas correctas

Descripción: Creamos un nuevo mapa en ETiles introduciendo medidas que sean válidas.

Condiciones de ejecución: Se ha ejecutado la aplicación.

Pasos seguidos:

1. A partir de menú, hacemos click sobre crear un nuevo mapa.
2. Introducimos los valores de anchura y altura del patrón del tile, y los tiles a lo largo y ancho que forman el mapa.
3. Pulsamos el botón **Aceptar**.

Resultado esperado: El mapa se debe crear con las dimensiones establecidas por el usuario.

Evaluación de la prueba: Realizada y satisfactoria.

7.3.2. Caso de prueba: Crear un nuevo mapa con medidas incorrectas

Descripción: Creamos un nuevo mapa en ETiles introduciendo medidas que no sean válidas.

Condiciones de ejecución: Se ha ejecutado la aplicación.

Pasos seguidos:

1. A partir de menú, hacemos click sobre crear un nuevo mapa.
2. Introducimos valores no numéricos en alguno de los campos, y en otros no.
3. Pulsamos el botón **Aceptar**.

Resultado esperado: El mapa se debe crear con las dimensiones establecidas por el usuario. En caso de que algún valor no haya sido correcto, se tomará el valor cero por defecto.

Evaluación de la prueba: Realizada y satisfactoria.

7.3.3. Caso de prueba: Actualización de la superficie del mapa al crearse

Descripción: Se comprueba que la superficie del mapa se pinta correctamente cuando se crea un nuevo mapa, apareciendo el scroll oportuno ante medidas que excedan la ventana del editor.

Condiciones de ejecución: Se ha ejecutado la aplicación.

Pasos seguidos:

1. A partir de menú, hacemos click sobre crear un nuevo mapa.
2. Introducimos los valores de anchura y altura del patrón del tile, y los tiles a lo largo y ancho que forman el mapa, cuyo resultado exceda las dimensiones de la ventana del editor.
3. Pulsamos el botón **Aceptar**.

Resultado esperado: Aparecen las ventanas de scroll, pudiendo desplazarnos a través de ellas, y se pinta la superficie del mapa del color de fondo.

Evaluación de la prueba: Realizada y satisfactoria.

7.4. Incremento 3: Creación de la lista de tiles

7.4.1. Caso de prueba: Insertar tiles

Descripción: Vamos a insertar varias imágenes a la lista tile a través de las tres opciones que tenemos, observando que se muestren correctamente.

Condiciones de ejecución: Se encuentra un mapa en edición actualmente.

Pasos seguidos:

1. Accedemos al diálogo de inserción haciendo click sobre el icono de insertar tiles en la barra de herramientas de la lista tiles, en el menú **Archivo->Importar** y a través del acceso rápido **Ctrl+I**.

2. Seleccionamos el fichero que contenga la imagen del tile.

3. Pulsamos el botón **Abrir**.

4. Repetimos los pasos anteriores varias iteraciones.

5. Seleccionamos cada uno de los tiles.

Resultado esperado: Las imágenes se pintan en la lista correctamente, con el tamaño indicado, por tanto, si el tamaño no coincide es escalado. Existe una separación entre cada tile, y pueden seleccionarse. Todos los enlaces que llevan al diálogo de inserción funcionan según lo esperado.

Evaluación de la prueba: Realizada y satisfactoria.

7.4.2. Caso de prueba: Eliminar tiles

Descripción: Vamos a eliminar varios tiles de la lista de tiles a través de las opciones que tenemos.

Condiciones de ejecución: Se encuentra un mapa en edición actualmente, y existen tiles en la lista de tiles.

Pasos seguidos:

1. Seleccionamos un tile.

2. Eliminamos el tile haciendo click sobre el icono de eliminar tiles en la barra de herramientas de la lista tiles.

3. Repetimos los pasos anteriores varias iteraciones.

Resultado esperado: Los tiles seleccionados se eliminan de la lista tiles, actualizándose el contenido de ésta como consecuencia. En el momento de eliminar, si existe datos en el mapa que se correspondan con el tile eliminado, también serán borrados.

Evaluación de la prueba: Realizada y satisfactoria.

7.4.3. Caso de prueba: Cargar tileset

Descripción: Se cargará un tileset, comprobando que se añade a la lista de tiles.

Condiciones de ejecución: Se encuentra un mapa en edición actualmente.

Pasos seguidos:

1. Hacemos click sobre el icono de añadir tileset en la barra de herramientas de la lista tiles.
2. Hacemos click sobre el botón **Examinar**.
3. Seleccionamos el fichero que queremos añadir.
4. Pulsamos el botón **Abrir**
5. Introducimos la separación entre cada tile.
6. Pulsamos el botón **Aceptar**

Resultado esperado: El diálogo se abre correctamente al pulsar sobre la herramienta de cargar tileset. La imagen seleccionada, se divide en los tiles que contiene según el patrón de tile que se encuentra en edición y con la separación indicada.

Evaluación de la prueba: Realizada y satisfactoria.

7.5. Incremento 4: Interacción entre el mapa y la lista de tiles

7.5.1. Caso de prueba: Añadir tiles al mapa

Descripción: Para probar la edición del mapa, añadimos a éste tiles que se encuentren en la lista de tiles, comprobando que se pinten en la posición indicada.

Condiciones de ejecución: Se encuentra un mapa en edición actualmente, y existen tiles en la lista de tiles.

Pasos seguidos:

1. Seleccionamos un tile de la lista tiles.
2. Movemos el puntero del ratón hacia la posición del mapa donde añadiremos el tile.
3. Hacemos click con el ratón.
4. Repetimos en proceso en distintas posiciones.

Resultado esperado: Durante el desplazamiento del ratón, se ilumina el tile donde se posiciona el cursor, además, se muestra sus coordenadas en la barra de estado. Se pinta el tile seleccionado en la posición indicada.

Evaluación de la prueba: Realizada y satisfactoria.

7.5.2. Caso de prueba: Mover los tiles de posición

Descripción: En esta prueba cambiamos de posición los distintos tiles que forman el mapa, comprobando que su funcionamiento sea el esperado.

Condiciones de ejecución: Se encuentra un mapa en edición actualmente, existen tiles en la lista de tiles y en el mapa.

Pasos seguidos:

1. Seleccionamos un tile del mapa.
2. Arrastramos el puntero del ratón hacia la nueva posición correcta.
3. Soltamos el botón del ratón.

Resultado esperado: Durante el desplazamiento del ratón, se ilumina el tile donde se posiciona el cursor, además, se muestra sus coordenadas en la barra de estado y una pequeña imagen escalada del tile que estamos moviendo junto al cursor. Al iniciar el movimiento, se elimina el tile de la posición, hasta completarse, que es cuando se pinta correctamente en la nueva posición.

Evaluación de la prueba: Realizada y satisfactoria.

7.6. Incremento 5: Barra de herramientas

7.6.1. Caso de prueba: Inicialización de la barra de herramientas

Descripción: Comprobamos que la barra de herramientas se muestra en la ventana de edición del editor, y se inicializa correctamente.

Condiciones de ejecución: Ninguna.

Pasos seguidos:

1. Ejecutamos la aplicación.
2. Comprobamos todos los enlaces haciendo click en cada uno de ellos.

Resultado esperado: Al ejecutar la aplicación se crea la ventana del editor, en la que se encuentra la barra de herramientas. Ésta de visualiza en la parte superior del editor, y está formada por los iconos indicados. Además, ninguno de ellos debe funcionar, excepto el creación y cargado de mapas.

Evaluación de la prueba: Realizada y satisfactoria.

7.7. Incremento 6: Herramienta de relleno y borrado

7.7.1. Caso de prueba: Rellenar mapa

Descripción: Usamos la herramienta rellenar para pintar el mapa que se encuentra en edición.

Condiciones de ejecución: Se encuentra un mapa en edición actualmente, y existen tiles en la lista de tiles.

Pasos seguidos:

1. Activamos la herramienta rellenar en la barra de herramientas.
2. Seleccionamos un tile de la lista tiles.
3. Movemos el puntero del ratón hacia la posición del mapa donde añadiremos el tile.
4. Hacemos click con el ratón.
5. Arrastramos hacia distintas posiciones.

Resultado esperado: La herramienta se activa correctamente. El tile seleccionado se pinta en la posición indicada por el cursor. Cuando arrastramos, se van pintando los tiles por los que el cursor pasó, si la posición estaba vacía.

Evaluación de la prueba: Se comprobó que al arrastrar hacia posiciones que excedieran los límites de la ventana de edición se accedía a posiciones inexistentes en la matriz de datos, por lo que es necesario comprobar que la posición del cursor es siempre correcta. Hecho el cambio y comprobado, la evaluación pasó a un estado satisfactorio.

7.8. Incremento 7: Guardar y cargar

7.8.1. Caso de prueba: Guardar el mapa en edición

Descripción: Comprobamos que la herramienta de guardado funciona correctamente para extensiones de tipo .map.

Condiciones de ejecución: Se encuentra un mapa en edición actualmente.

Pasos seguidos:

1. Activamos el guardado de mapa en el menú **Archivo->Guardar Como** o pulsando **Ctrl+Shift+S**.
2. Introducimos el nombre con el que vamos a guardar el mapa, seguido de la extensión .map.
3. Seleccionamos la carpeta donde se guardará el fichero.
4. Pulsamos el botón **Aceptar**.

Resultado esperado: El diálogo de guardado de mapa se abre correctamente. El mapa se guarda correctamente en la carpeta seleccionada y con el nombre especificado.

Evaluación de la prueba: Realizada y satisfactoria.

7.8.2. Caso de prueba: Exportar el mapa a distintos formatos

Descripción: Probamos que el mapa se puede exportar a todos los formatos, es decir, el formato para bibliotecas .mbp y los formatos típicos para imágenes.

Condiciones de ejecución: Se encuentra un mapa en edición actualmente.

Pasos seguidos:

1. Activamos el exportado de mapa en el menú **Archivo->Exportar** o pulsando **Ctrl+E**.
2. Introducimos el nombre con el que vamos a exportar el mapa, seguido de la extensión.
3. Seleccionamos la carpeta donde se guardará el fichero.
4. Pulsamos el botón **Aceptar**.
5. Repetimos el proceso para los distintos formatos.

Resultado esperado: El diálogo de exportación se abre correctamente. El mapa se guarda en la carpeta seleccionada y con el formato especificado. Si el formato no se reconoce, se muestra el error y se cancela la exportación.

Evaluación de la prueba: Realizada y satisfactoria.

7.9. Incremento 8: Herramientas selección, copiar, cortar y pegar

7.9.1. Caso de prueba: Seleccionar una región del mapa

Descripción: Utilizamos la herramienta seleccionar para comprobar que el comportamiento es el esperado.

Condiciones de ejecución: Se encuentra un mapa en edición actualmente.

Pasos seguidos:

1. Activamos la herramienta seleccionar en la barra de herramientas.
2. Hacemos click sobre la posición del primer tile que vamos a seleccionar.
3. Arrastramos el ratón hacia nuevas posiciones.
4. Repetimos el proceso desde distintas posiciones.

Resultado esperado: La herramienta se activa correctamente. El primer tile seleccionado se resalta en primera instancia. Al arrastrar, se une el tile de comienzo con del final formando un rectángulo, convirtiéndose en la región seleccionada. Si la posición de arrastre excede la ventana de edición, queda en pantalla el último rectángulo seleccionado.

Evaluación de la prueba: Realizada y satisfactoria.

7.9.2. Caso de prueba: Pegar selección

Descripción: Probamos la herramienta de pegado sobre el mapa una vez hayamos copiado una región.

Condiciones de ejecución: Se encuentra un mapa en edición actualmente, y se ha copiado una región.

Pasos seguidos:

1. Activamos la herramienta de pegado a partir del menú **Editar->Pegar** o pulsando **Ctrl+V**.

2. Movemos el rectángulo de pegado hacia la posición donde queremos realizar el pegado.

3. Hacemos click con el ratón.

4. Repetimos en distintas posiciones.

Resultado esperado: La herramienta se activa correctamente. El rectángulo de pegado se resalta correctamente sobre la posición del cursor. Al hacer click, se pega el contenido del rectángulo siempre y cuando dicha posición no se encuentre ocupada. Si la nueva posición excede las dimensiones del mapa, sólo se pega la zona que se encuentra dentro del mapa.

Evaluación de la prueba: Realizada y satisfactoria.

7.10. Incremento 9: Herramientas zoom, deshacer y rehacer

7.10.1. Caso de prueba: Ampliar y reducir el zoom del mapa

Descripción: Usamos la herramienta de ampliado y reducción de mapa para comprobar que su funcionamiento es según lo esperado.

Condiciones de ejecución: Se encuentra un mapa en edición actualmente.

Pasos seguidos:

1. Activamos la herramienta de ampliado a través del icono de la barra herramientas o pulsando **Ctrl++**, hasta llegar al zoom máximo.

2. Activamos la herramienta de reducción a través del icono de la barra de herramientas o pulsando **Ctrl+-**, hasta llegar al zoom mínimo.

3. Repetir el primer paso hasta llegar al zoom inicial.

Resultado esperado: Las herramientas se activan correctamente para cada uno de sus enlaces. Cuando se aumenta y disminuye el zoom, cada tile que compone el mapa se actualiza correctamente, apareciendo y desapareciendo el scroll necesario en cada momento.

Evaluación de la prueba: Realizada y satisfactoria.

7.10.2. Caso de prueba: Deshacer acciones

Descripción: En esta ocasión utilizamos la herramienta deshacer para comprobar que se pueden recuperar estados anteriores de nuestro mapa.

Condiciones de ejecución: Se encuentra un mapa en edición actualmente, y se han realizado todas las diferentes acciones que el editor permite.

Pasos seguidos:

1. Activamos la herramienta deshacer a través de la barra de herramientas y la tecla de acceso rápido **Ctrl+Z**.
2. Repetimos el proceso.

Resultado esperado: La herramienta se activa para las dos opciones disponibles. Para cada acción se recupera el estado anterior, ya sea una operación de inserción, eliminación o movimiento, todo en el orden correcto.

Evaluación de la prueba: Realizada y satisfactoria.

7.11. Incremento 10: Herramienta de redimensión y propiedades

7.11.1. Caso de prueba: Redimensionar el mapa a un tamaño menor

Descripción: En esta prueba, comprobamos que el comportamiento de la herramienta redimensionar es el correcto cuando se redimensiona el mapa a un tamaño menor.

Condiciones de ejecución: Se encuentra un mapa en edición actualmente.

Pasos seguidos:

1. Activamos la herramienta redimensionar a través del menú **Propiedades->Redimensionar Mapa** o pulsando **Ctrl+M**
2. Insertamos las nuevas medidas en el diálogo de redimensión.
3. Pulsamos el botón **Aceptar**.

Resultado esperado: La herramienta se activa correctamente, apareciendo el diálogo de redimensión con las medidas del mapa actual. Cuando se ejecuta se redimensiona el mapa a las nuevas medidas copiando solo los tiles que tienen cavidad dentro del nuevo mapa.

Evaluación de la prueba: Realizada y satisfactoria.

7.12. Incremento 11: Mapas aleatorios

7.12.1. Caso de prueba: Creación de un mapa aleatorio

Descripción: Utilizamos la herramienta de creación de mapas aleatorios para crear el mapa de forma automática, comprobando que su funcionamiento es el correcto.

Condiciones de ejecución: La aplicación se ha ejecutado.

Pasos seguidos:

1. Activamos la herramienta accediendo al menú **Archivo->Nuevo->Aleatorio** o pulsando **Ctrl+A**.
2. Introducimos las dimensiones del mapa.
3. Seleccionamos el algoritmo.
4. Insertamos los tiles correspondientes que formarán el mapa.
5. Indicamos la composición del mapa.
6. Pulsamos el botón **Aceptar**.

Resultado esperado: El diálogo de creación de mapas aleatorios se abre correctamente. Éste permite introducir las dimensiones del mapa, seleccionar un solo algoritmo entre los que hay, y mover las barras de desplazamiento para indicar la composición del mapa. Al terminar, el mapa se crea y se pinta correctamente con todo lo indicado.

Evaluación de la prueba: Realizada y satisfactoria.

7.13. Incremento 12: Minimapa

7.13.1. Caso de prueba: Desplazar a partir del minimapa

Descripción: Usamos el minimapa de la aplicación para trasladar la visión del mapa a la posición que indiquemos en el minimapa.

Condiciones de ejecución: Se encuentra un mapa en edición actualmente.

Pasos seguidos:

1. Hacemos click sobre una posición del minimapa.
2. Repetimos el proceso para distintas posiciones.

Resultado esperado: Al hacer click sobre el minimapa, la ventana de visión del mapa cambia, situándose la esquina superior izquierda sobre el punto presionado sobre el minimapa.

Evaluación de la prueba: Realizada y satisfactoria.

7.14. Incremento 13: Biblioteca libSDL

Éste es el único apartado en el que las pruebas han sido realizadas por mi, ya que los usuarios que han probado el editor no poseían conocimientos de programación.

Durante las pruebas se creaban pequeños programas de prueba, los cuales se adjuntan en el proyecto, que trabajan con la biblioteca libSDL, y que comprueba que cada función se ejecutaba correctamente y sin fallos.

Capítulo 8

Conclusiones

8.1. Análisis del resultado

Llegado a este punto debemos realizar un pequeño resumen sobre las conclusiones que sacamos después de haber realizado tanto esfuerzo para realizar el proyecto ETiles.

En primer lugar, hay que destacar que el proyecto aún puede sufrir numerosas expansiones que lo conviertan en un producto mucho más completo, capaz de llegar a un número más amplio de usuarios. Por tanto, desde aquí se anima a toda persona que quiera colaborar con el proyecto, ya que este proyecto es realmente gratificante y rápidamente se obtienen los frutos de tantas horas de dedicación y entrega a la elaboración del mismo. Por ello se adjunta a modo de apéndice un pequeño manual que con unas pocas indicaciones orienta al programador como añadir nuevas herramientas.

En cuanto a la calidad obtenida en el proyecto creo que los resultados han sido bastantes satisfactorios, ya que aún siendo mi primera aplicación creada bajo la biblioteca Qt, ETiles es un proyecto sólido, que ofrece todas las funcionalidades necesarias para crear mapas de calidad y sin errores.

Aparte de lo aportado personalmente por la realización del editor, me encuentro también satisfecho con todo lo aprendido y con todas las herramientas que desconocía y he llegado a aprender con el fin de elaborar el proyecto con la mayor calidad posible.

Comenzaré con la herramienta *GIMP*, que ha sido la utilizada para diseñar los diferentes tiles de cosecha propia, ya que es una herramienta muy completa y una vez que aprendes a usarlo es muy útil para cualquier aspecto gráfico.

Otra de las herramientas utilizadas ha sido *Qt Creator*, que ha proporcionado un perfecto entorno de desarrollo integrado para programar en C++ junto con las librerías de desarrollo Qt. Entre las características a destacas encontramos la posibilidad de tener controlados todos los ficheros que forman parte de la aplicación, ayudándonos en la creación de código repetitivo, auto-completado de métodos y variables, sistema de resaltado de errores de sintaxis, etc.

Como se ha podido observar durante todo el proyecto, ETiles está creado con las clases que ofrece la biblioteca de desarrollo Qt. La razón por la cual hemos escogido esta biblioteca entre todas las disponibles ha sido principalmente por la excelente documentación que presenta, que nos ha ayudado a solucionar todas las dudas que nos han surgido, además de por su enorme potencia para desarrollar entornos gráficos y facilidad de uso. Por todo esto, estoy muy contento de haber conseguido un nivel bastante alto en el uso de este tipo de librerías.

Hemos usado también la librería **libSDL**. Cómo se ha comentado ya, es una biblioteca que se encuentra en auge actualmente en el desarrollo de videojuegos. Por tanto, era una buena elección sobre la que desarrollar nuestra propia librería.

Y algunas herramientas más que hemos utilizado son; Subversion, para llevar un control de versiones de la aplicación y subir el contenido a la forja donde se encuentra el proyecto alojado. Doxygen, para documentar todo el código fuente, y Planner, que nos ha ayudado a planificar todo el desarrollo de ETiles, aprendiendo a gestionar el tiempo de manera productiva.

Respecto al proceso de desarrollo software, he aprendido a aplicar una de las metodologías de desarrollo de la Ingeniería de Software estudiadas en la carrera. Decir, que me ayudó a llevar un control de cada una de las etapas que debía realizar y que siguiendo cada una de las indicaciones que nos ofrece dicha metodología se han cumplido con los objetivos inicialmente propuestos.

Por último, destacar que se aprende que cualquier persona aunque no tenga conocimientos informáticos puede ayudarte en el desarrollo de tu proyecto. Por ello hay que saber escuchar a las personas, saber tratarlas y respetar sus opiniones, ya que a veces podemos no observar qué es lo que realmente nos está pidiendo el usuario, equivocándonos en las funcionalidades que debe prestar la aplicación.

8.2. Posibles ampliaciones

En futuras versiones el proyecto podría tener una expansión bastante importante, a continuación se muestran alguna de las numerosas ideas que se podría implantar. Serían por orden de prioridad:

1. Desarrollar la perspectiva isométrica. Los mapas completamente en 2D no son los únicos que se pueden desarrollar mediante las unión de tiles. Implementar esta perspectiva constituiría una representación de los mapas tridimensional en dos dimensiones.
2. Desarrollar los mapas hexagonales. Se podría ampliar el proyecto para que aceptara crear mapas con tiles hexagonales como base, ya que muchas veces los desarrolladores de videojuegos apuestan por este tipo de mapas.
3. Creación de un portal web. Podría ser interesante crear un sitio web donde una comunidad de desarrolladores pudieran comunicarse y trabajar juntos para mejorar el proyecto y compartir sus mapas.

Apéndice A

Manual de usuario

En el siguiente manual explicaremos con detalle como usar el editor de forma eficiente, describiendo cada herramienta y las acciones que se pueden llevar a cabo con ellas, llegando a resolver cualquier posible duda que pueda tener el usuario.

A.1. Ejecución

Una vez instalado, tenemos que ejecutar el editor escribiendo en consola:

```
./etiles
```

Nos aparecerá la ventana principal del programa.

Ver figura 3.9

A.2. Primeros pasos

La ventana principal se compone de cuatro zonas principales:

- La zona central, donde se mostrará el mapa una vez haya sido creado.
- La zona superior derecha, donde se encuentra el minimapa.
- La zona inferior derecha, donde se encuentra la lista que componen los tiles que usaremos para crear nuestro mapa.
- La zona superior, formada por los menús del editor y la barra de herramientas.

Si es la primera vez que ejecuta ETiles desde su instalación, lo primero que deberá hacer es crear un nuevo mapa. Para ello tenemos dos opciones, crear un mapa vació de las dimensiones que desee o usar la herramienta de creación de mapas aleatorios que proporciona el editor.

A.2.1. Crear un mapa normal

Para crear un mapa normal puede usar la barra de herramientas, el menú **Archivo->Nuevo->Normal**, o pulsar **Ctrl+N**. Use la opción que use le aparecerá una ventana como la siguiente:

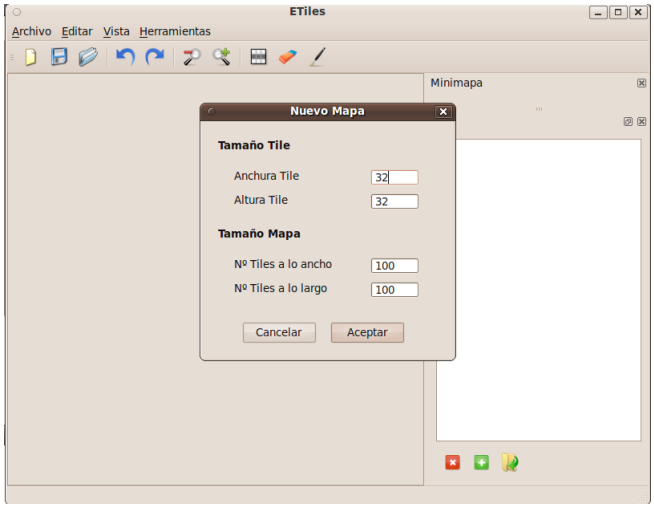

Figura A.1: Diálogo crear nuevo mapa normal

Sólo debe especificar la anchura y la altura del patrón de tile que compone el mapa, y el número de ellos a lo ancho y largo. Por ejemplo, si decidiera un tamaño de 32x32 y una cantidad de 100x100 tiles, el tamaño del mapa será igual 3200x3200 píxeles. Si está seguro de las medidas pulse en botón **Aceptar**.

A.2.2. Crear un mapa aleatorio

Para crear un mapa aleatorio puede usar el menú **Archivo->Nuevo->Aleatorio**, o pulsar **Ctrl+A**. Inmediatamente aparecerá una nueva ventana parecida a la resultante de seleccionar los mapas normales pero con más opciones.

Lo primero que tiene que hacer es indicar las dimensiones de su mapa como si se tratará de un mapa normal. Ya que si cargamos un tileset puede que no cargue los tiles con las medidas con las que quiere y puede producir resultados no esperados.

Una de la nuevas opciones que aparece es una lista tiles pero de menor tamaño, de idéntico uso a la que posee el editor, donde el usuario puede insertar los tiles de los que quiere que su mapa se componga.

Por último, solo debe de indicar las características que quiere que posea su mapa a través de las barras de movimiento y seleccionar el algoritmo deseado. Podemos observar un ejemplo de uso en las siguiente imágenes:

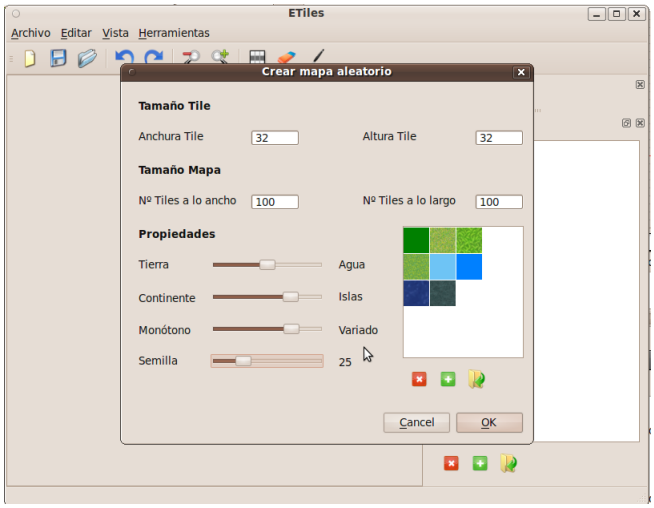

Figura A.2: Diálogo de creación de mapas aleatorios

Que producirá la siguiente salida:

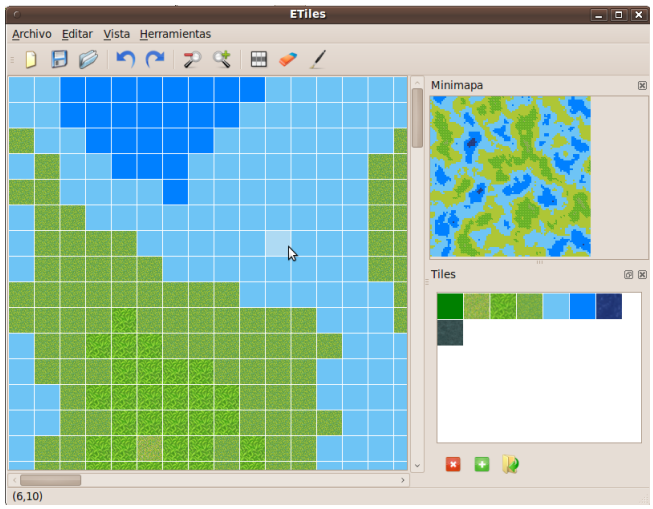

Figura A.3: Mapa aleatorio creado en ETiles

A.2.3. Guardar mapa

Ya creado el mapa, en caso de querer modificarlo posteriormente deberá guardarlo. Al pulsar el botón guardar de la barra de herramientas, el menú **Archivo->Guardar** o **Ctrl+S**, si es la primera vez, actuará como la herramienta guardar como(**Archivo->Guardar Como** o **Ctrl+Shift+S**) , y se abrirá directamente el diálogo de guardado que le permitirá escribir el nombre que elija para su mapa, añadiendo la extensión **.map**, que es el formato que usa el editor. Si ya hubiese sido guardado se guardará automáticamente a menos que especifiquemos usar guardar como.

Además del formato específico del editor también puede guardar el mapa como una imagen, o con el formato .mbp usado para la librería libSDL. Para ello sólo debe acceder al diálogo de exportar a partir

del menú **Archivo->Exportar** o **Ctrl+E**, indicando el nombre con el que desea guardar el fichero y la extensión.

A.2.4. Cargar mapa

Si no es la primera vez que ejecuta ETiles, y anteriormente ha creado un mapa que ha guardado, puede usar está opción para recuperarlo y continuar el trabajo.

Tiene varias opciones, usar el menú **Archivo->Cargar**, pulsar el icono cargar de la barra de herramientas o pulsar **Ctrl+L**. Hecho ésto, aparecerá el diálogo de cargado que le permitirá escoger el archivo con extensión **.map** previamente guardado en otra sesión.

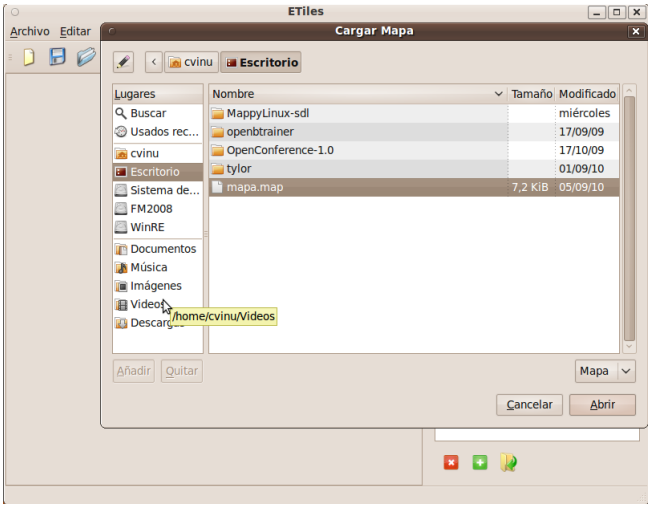

Figura A.4: Diálogo cargar mapa

Puede ocurrir que exista un mapa en edición en el momento de realizar el cargado que ha sido modificado desde la última vez que se guardó. Si ésto ocurre el editor posibilita la opción de guardarlos antes de continuar.

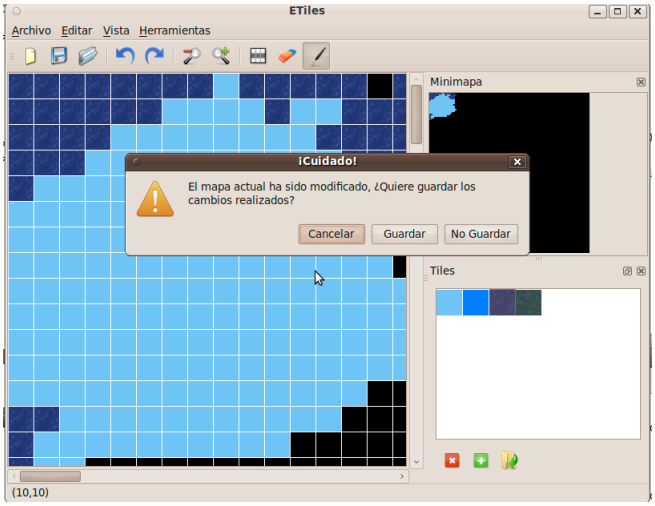

Figura A.5: Mensaje de confirmación

A.3. Edición de ETiles

A continuación pasaremos a explicar las diferentes formas de que disponemos para editar nuestro mapa y cada una de las herramientas que nos ofrece ETiles.

A.3.1. Añadir tile

Comenzamos con la edición del mapa, para ello en principio sólo es necesario usar el cursor del ratón y tener las imágenes de los tiles cargadas en la lista de tiles. Seleccione el tile que desee de dicha lista, éste quedará marcado como seleccionado, a continuación haciendo click sobre la posición del mapa conseguirá que el tile se pinte sobre el mapa.

Por comodidad, cuando se mueve el cursor sobre el mapa se iluminará la casilla correspondiente del tile al que pertenece, además en la barra de estado inferior de la ventana de edición se mostrarán las coordenadas (x,y).

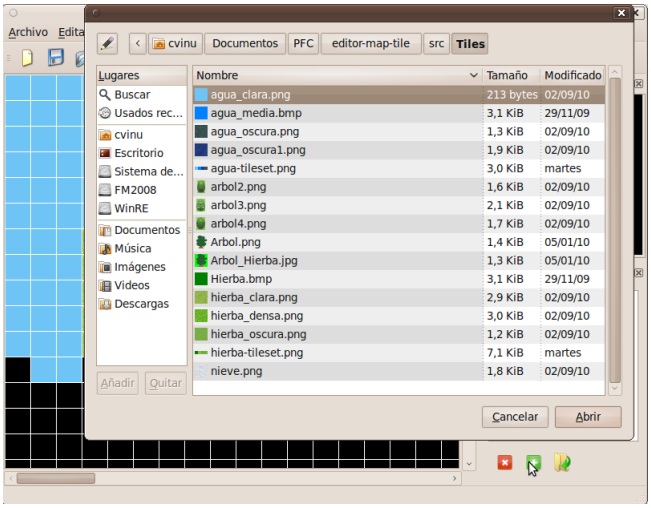

Figura A.6: Añadir tile

A.3.2. Mover tile

Cuando hay al menos un tile añadido en el mapa puede cambiarlo de posición haciendo click sobre él, y sin soltar el botón del ratón, desplace hacia la nueva posición y suéltelo. Cuando esté moviendo el ratón observará que la imagen escalada del tile aparecerá junto al cursor. Si la nueva posición no está dentro de los límites del mapa, el movimiento no se llevará a cabo y se reseteará el tile en la antigua posición.

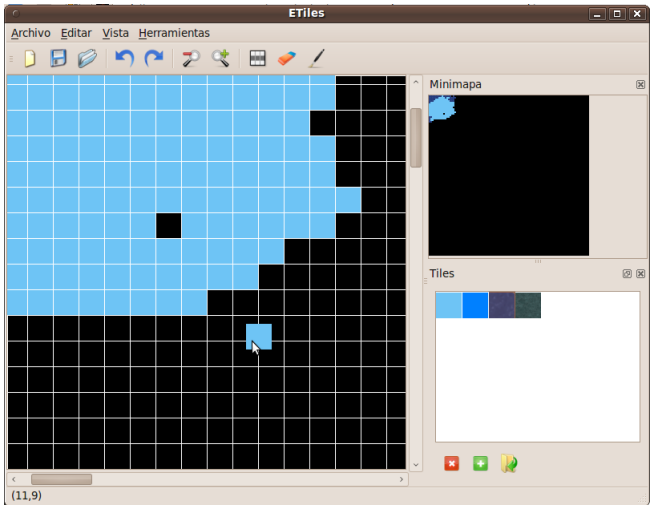

Figura A.7: Movimiento de un tile por la ventana de edición del mapa

A.3.3. Eliminar tile

Para eliminar un tile de nuestro mapa hay que seleccionar la herramienta de borrado que se encuentra en la barra de herramientas. Hecho esto, haciendo click sobre sobre la posición del tile, siempre y cuando exista, será eliminado. Si quiere una mayor velocidad de borrado sólo debe mover el ratón una vez hecho click, y sin soltar el botón del ratón, arrastrar hacia todas las posiciones que vaya a eliminar.

Una segunda forma para eliminar es utilizar la herramienta de selección que veremos más adelante.

A.3.4. Herramienta rellenar

Se encuentra en la barra de herramientas y sólo debe seleccionarla para usarla. Su funcionamiento es el mismo que para la herramienta de borrado, cuando tenga un tile seleccionado en la lista de tiles, haga click en la posición donde quiera colocarlo, y arrástrelo en caso de querer pintarlo en más de una posición.

El editor nos permite mover los tiles cuando el relleno está activado, al contrario que con el borrado y la selección.

A.3.5. Herramienta seleccionar

Al igual que el resto, se encuentra en la barra de herramientas. Para usarla selecciónela y haga click en el mapa sobre el tile que prefiera, el cual se sombreará para indicarlo. Si una vez hecho click, arrastramos hacia una nueva posición, la selección tomará forma rectangular uniendo el tile de comienzo y el tile de finalización.

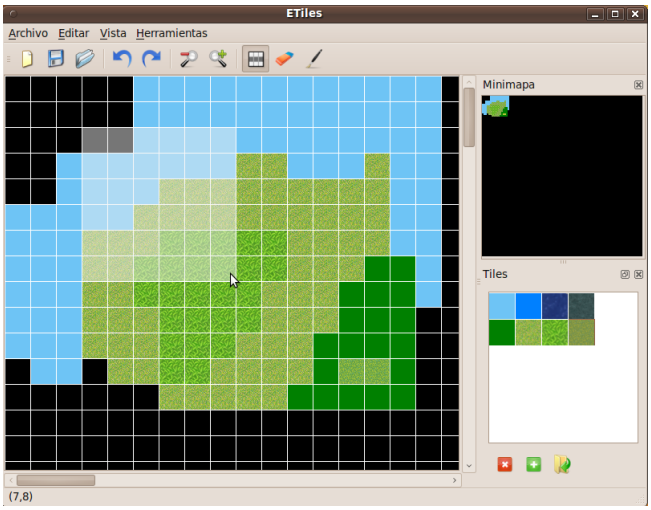

Figura A.8: Herramienta selección

La selección solo se llevará a cabo si la posición es siempre correcta, en caso contrario, no. Con esta herramienta podrá utilizar otras como copiar,cortar, pegar o eliminar la zona seleccionada.

A.3.6. Herramientas copiar y cortar

Seguimos con las típicas herramientas de edición, como son copiar y cortar, que junto a pegar pueden ser de mucha utilidad. Para un correcto uso previamente debes haber seleccionado algún tile del mapa. Luego puede usar el menú **Editar** o **Ctrl+C** para copiar y **Ctrl+X** para cortar.

Cuando use cortar tiene que tener en cuenta que solo los tiles que existan dentro del rectángulo de selección serán copiados, el resto serán tiles vacíos. Además de evidentemente ser eliminados, al contrario que con copiar.

A.3.7. Herramienta pegar

La siguiente herramienta que veremos será pegar. Como es lógico es necesario haber usado antes las herramientas de copiado o cortado. Si todo es correcto puede pulsar **Ctrl+V** o dirigirse al menú **Edición** para llevar a cabo el pegado. Consecuentemente, si dirige el cursor hacía una posición válida dentro del mapa, observará cómo se sombreará el rectángulo que corresponde con el rectángulo copiado o cortado previamente. Por tanto, a medida que mueve el ratón este rectángulo se desplazará hasta que encuentre la posición donde desee realizar el pegado, teniendo que hacer click para que se lleve a cabo.

Recuerde que el tile superior izquierda del rectángulo de pegado es el que se posicionará en el tile que refleja el cursor en el mapa, y por tanto, si se encuentra en los tiles del borde, no se realizará el pegado completo, y sólo se pegarán los tiles que tengan cavidad.

A.3.8. Herramienta zoom

No puede faltar la herramienta zoom en un editor, y como en todos, puede aumentar o disminuir el zoom del mapa para una mayor comodidad usando el menú **Vista** o a través de las teclas de acceso rápido **Ctrl++** y **Ctrl+-**.

Los posibles zoom's permitidos son los siguientes: 20 %,50 %,75 %,100 %,125 %,150 %,200 %.

A.3.9. Herramientas rehacer y deshacer

Cuando queramos recuperar un estado anterior o posterior en nuestro mapa, el editor ofrece estas dos herramientas que nos permite hacerlo de forma rápida y sencilla. Estas opciones se encuentran en el menú **Editar** o pulsando **Ctrl+Z** para deshacer y **Ctrl+Shift+Z** para rehacer.

Por ejemplo, si elimina un tile de forma accidental o realiza un movimiento indebido puede recuperar el estado anterior usando la acción deshacer.

Actualmente estas dos opciones sólo funcionan con la edición del mapa, y no con la lista de tiles, por tanto, tenga cuidado a la hora de borrar o añadir tiles de dicha lista porque deberá recuperarlos de forma manual.

A.3.10. Redimensionar mapa

Puede ocurrir alguna vez que hayamos creado un mapa y comenzado la edición, para más tarde tener la necesidad de redimensionar las medidas de nuestro mapa, teniendo que crear uno nuevo con distintas dimensiones y perder el trabajo realizado. Por ello, esta herramienta se hace indispensable para el editor ofreciendo poder realizar esta reestructuración sin perder nada de lo trabajado. Tiene que pulsar el menú **Herramientas** ó **Ctrl+R**, y le aparecerá una ventana como la siguiente:

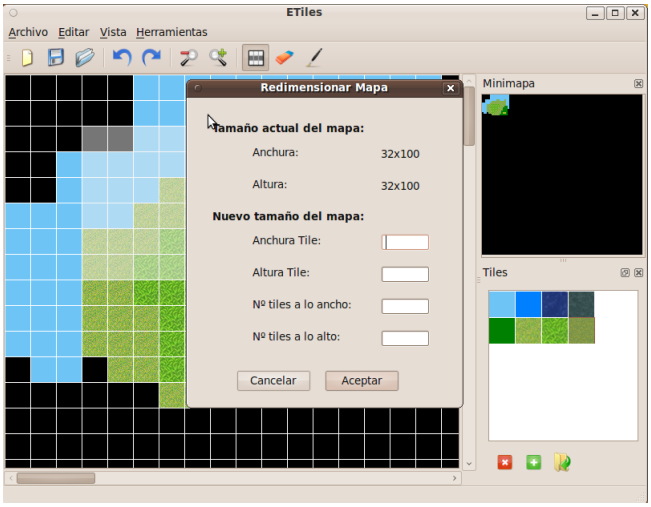

Figura A.9: Dialogo redimensionar mapa

Como puede observar se parece mucho al diálogo **Crear nuevo mapa normal** A.1, con la única diferencia que nos muestra las medidas actuales.

Tenga en cuenta, que si las nuevas medidas son más pequeñas que las anteriores los tiles excedentes serán borrados.

A.3.11. Rejilla

La rejilla es una herramienta opcional que nos delimita vertical y horizontalmente cada tile del mapa para una mayor comodidad. Si no nos interesa esta opción podemos retirarla a través del menú **Vista** o pulsando la tecla **R**.

A.4. Lista de tiles

La lista de tiles es la parte del editor donde se guardan los tiles que unidos entre sí componen nuestro mapa final. A partir de ella podrá seleccionar el tile que vaya a usar para alguna de las herramientas de edición descritas anteriormente.

En un principio, la lista se visualizará abajo a la derecha del editor, pero si prefiere cualquier otro lugar puede desplazarla a la zona que prefiera haciendo click sobre ella y arrastrándola hacia otra posición. Incluso puede flotar la lista como una ventana independiente. Si cierra la ventana, puede volver a mostrarla a través del menú **Vista**.

A.4.1. Importar tile

Con esta acción podemos añadir tiles a la lista tiles a partir del menú **Archivo**, pulsando **Ctrl+I** o con el botón existente en la lista. En el momento de accionarlo se abrirá un nuevo diálogo permitiendo navegar por la carpetas de su sistema para buscar las imágenes que quiere insertar.

Puede elegir cualquier imagen con los formatos típicos de éstas, que se almacenará en la lista como un único tile.

Ver figura A.6

A.4.2. Cargar tileset

Nuestro mapa puede estar compuesto por muchos tiles distintos, e ir añadiéndolos uno a uno puede resultar una tarea costosa de tiempo y esfuerzo. Para solucionar este problema existen los llamados tileset, que no son mas que varios tiles unidos entre sí y que se guardan como una sola imagen. El editor permite cargarlos con el último botón que existe en la lista de tiles.

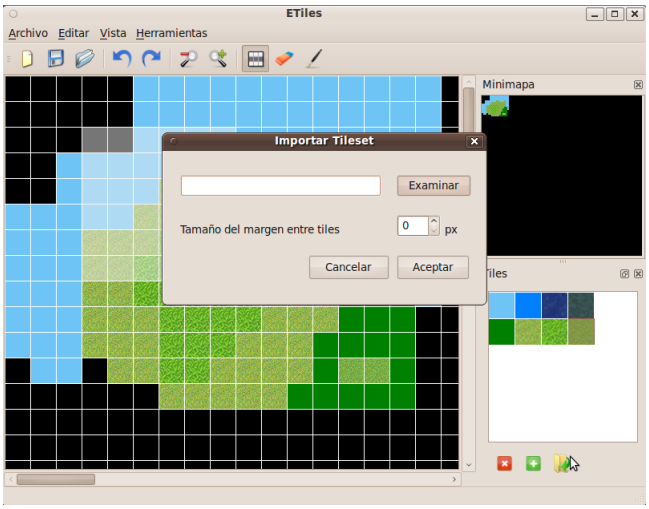

Figura A.10: Cargar tileset

Se abrirá un diálogo como el que se ha mostrado, en el que hay que indicar la ruta del archivo donde se encuentra el tileset. Si no lo sabe con exactitud puede cliquear el botón **Examinar** que le ayudará a encontrarlo. También debe indicar la separación en píxeles entre cada tile dentro de la imagen.

Recuerde que si el tamaño de cada tile que se añade a lista no coincide con el del patrón del mapa, se escalará el tamaño al del patrón.

A.4.3. Borrar tile

Cuando un tile insertado previamente deje de ser útil, puede seleccionarlo en la lista tile y eliminarlo pulsando el botón eliminar de susodicha lista.

Tenga cuidado a la hora de realizar ésta acción ya que si elimina un tile que está siendo usado en la edición del mapa, las posiciones que estén ocupadas por éste serán eliminadas perdiendo información. Además, si pertenece a un tileset, cada uno de los tiles que lo componían serán eliminados también, con las misma consecuencia. Para evitar errores de ETiles provee un diálogo de confirmación.

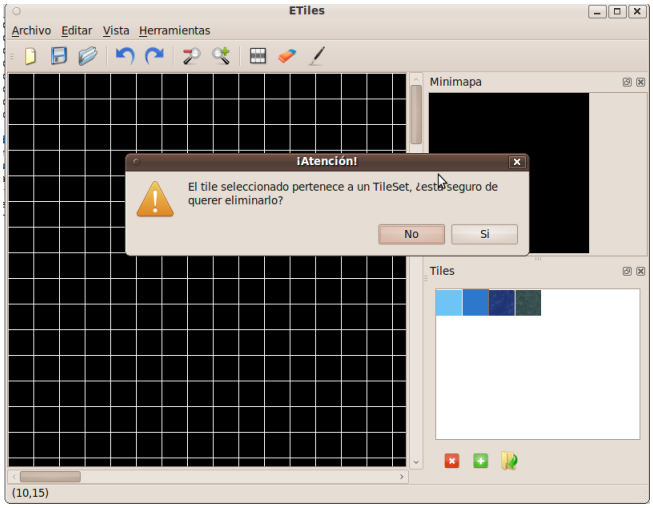

Figura A.11: Diálogo de confirmación de eliminación de un tileset

A.5. Minimapa

El minimapa nos puede ayudar mucho a la hora de realizar nuestro mapa ya que ofrece una visión completa a una escala reducida de éste. Se irá actualizando a medida que vayamos insertando o borrando tiles de nuestro mapa. Además, podemos hacer click sobre él, transportando la visión del mapa a la zona correspondiente, siempre y cuando, el tamaño del mapa exceda el tamaño de la ventana.

Si no nos interesa, podemos cerrarlo, evitando que se muestre o simplemente cambiarlo de posición.

A.6. Propiedades

Por el momento las propiedades a modificar en el editor son pocas. Si pulsamos el menú **Herramientas** o **Ctrl+O**, se abrirá un dialogo como el siguiente:

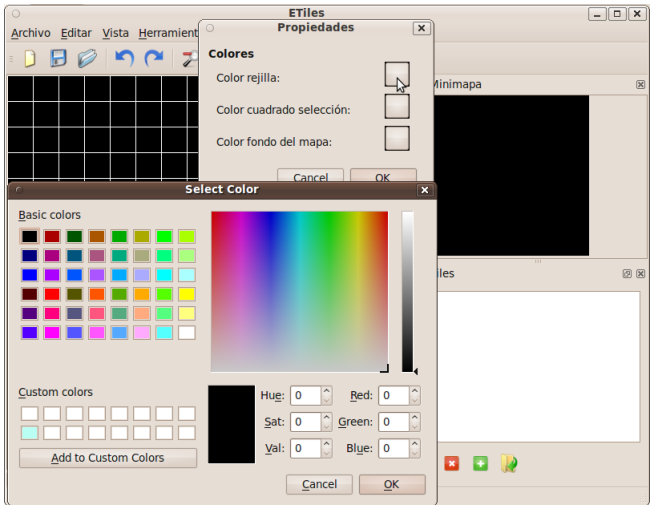

Figura A.12: Propiedades del mapa

Como observa en la imagen, puede modificar el color de la rejilla, el color de fondo y el color de selección por si le es necesario un mayor contraste. El uso es similar para los tres, pulse sobre el botón a la derecha del nombre y se abrirá un dialogo que le permitirá escoger el color que desee. Los cambios no se llevarán a cabo hasta que pulse el botón **Aceptar**.

A.7. Uso de la biblioteca libSDL

Este apartado va destinado a programadores que vayan a hacer uso de la biblioteca de ETiles para la librería libSDL, bajo el lenguaje C++

Lo primero que debe hacer es incluir la biblioteca al fichero donde vaya a hacer uso de ella. Seguidamente, declaramos una instancia de la clase con el constructor predeterminado.

```
1    SDLETiles mapa;
```

Creado el mapa vacío, tiene dos posibles caminos. Cargar un mapa que ha creado previamente usando el editor gráfico de ETiles y que ha exportado usando la extensión .mbp, indicando la ruta de dicho fichero y la ruta donde se encuentran las imágenes que uso para desarrollar el mapa.

```
1    mapa.cargarMapa(fichero,rutaTiles);
```

Todas las imágenes deben estar en la misma carpeta para que el cargado se realice correctamente.

Por otro lado, puede crear un nuevo mapa aleatorio.

```
1    mapa.crearMapaAleatorio(20,20,40,30,
2                            listatiles,82,44,86,6,0);
```

La lista de tiles contiene todas las imágenes de los tiles, que serán de tipo `SDL_Surface *`. Deben tener el mismo tamaño que el patrón del tile, ya que si no a la hora de pintar el mapa sólo se pintara la parte de la imagen que tenga cavidad en la posición que ocupa.

También puede pintar el mapa en la superficie que desee.

```
1  mapa.pintarEnSuperficie(superficie);
```

Con esta función conseguimos pintar el mapa en la superficie dada por parámetro. La superficie será de tipo `SDL_Surface *`, y para que el pintado se lleve a cabo correctamente debe tener el mismo tamaño que el tamaño del mapa, si no, sólo se pintara parcialmente. Para crear la superficie, puede hacer uso de la función `SDL_CreateRGBSurface()` de la biblioteca libSDL.

Apéndice B

Manual de instalación

B.1. Descargar ETiles

En primer lugar deberemos descargar la versión de ETiles. Para ello puedes acceder a la página donde se encuentra alojada [8] o usar `subversion` para descargarlo directamente desde el repositorio mediante la siguiente instrucción:

```
svn checkout https://forja.rediris.es/projects/editor-map-tile/
```

B.2. Instalación de la librería de desarrollo Qt

Para poder compilar ETiles es necesario tener instalada la librería Qt, por lo que si ya la tenemos instalada podemos saltarnos este paso.

Si deseamos obtener la última versión de la librería Qt, accedemos a la zona de descargas de su página oficial a partir del siguiente enlace:

Zona de descarga de página oficial de la librería Qt

Elegimos la opción LGPL, seguidamente seleccionamos el enlace de librería Qt para Linux y la descargamos.

Luego abrimos la consola, accedemos a la carpeta contenedora y descomprimimos el archivo:

```
cd Descargas
tar -xvzf qt-fichero.tar.gz
```

A continuación, accedemos a la carpeta descomprimida y realizamos la instalación:

```
cd qt-fichero
./configure
make
make install
```

Por último, debemos extender la variable PATH para poder compilar nuestros programas Qt, para ello debemos añadir las siguientes líneas en nuestro archivo .profile que se encuentra en nuestra carpeta personal.

PATH=/usr/local/Trolltech/QtEmbedded-4.7.0/bin:$PATH
export PATH

Si queremos instalar la librería desde los repositorios de la distribución, sólo debemos escribir la orden necesaria en la consola, seguido del paquete. Para distribuciones basadas en Debian:

```
sudo apt-get install libqt4-dev
```

B.3. Instalación ETiles

En caso de haber bajado el programa directamente desde la web, debemos descomprimir el archivo usando la siguiente instrucción:

```
tar -xvzf etiles-v1.0c.tar.gz
```

Realizada la descompresión si hemos decidido optar por la descarga directa, u obtenido los ficheros a través de subversion, accedemos a la carpeta contenedora del código:

```
cd etiles/src/
```

Si tenemos las librerías de desarrollo Qt instaladas, solo deberemos escribir:

```
qmake
make
```

Si la compilación ha sido satisfactoria nos generará un fichero ejecutable llamado etiles, el cual solo deberemos ejecutar para usar la aplicación.

```
./etiles
```

Apéndice C

Manual del desarrollador

El presente manual está indicado para que cualquier desarrollador con unas nociones de programación en C/C++, la librería de desarrollo Qt, y libSDL puedan desarrollar nuevas herramientas o mejorar la ya existentes.

C.1. Estructura de carpetas

ETiles posee una jerarquía muy clara de carpetas, para facilitar la ampliación o modificación de alguna funcionalidad a todo aquel desarrollador que lo desee. La aplicación posee tres carpetas principales, cada una de ellas con un contenido específico, que pasamos a comentar a continuación:

doc Es la carpeta en la que se incluye esta memoria y los fuentes LaTeX con la que se ha creado dicha memoria.

imagenes En esta carpeta nos encontramos con todas las imágenes que se incluyen en ETiles. Dentro de esta carpeta se incluyen otras dos.

- **tiles** En esta carpeta se incluyen todos los tiles de prueba del proyecto.
- **iconos** Aquí se incluyen las imágenes de los iconos que usaremos para las barras de herramientas.

src Esta carpeta incluye todos los ficheros fuente del proyecto.

C.2. Creación de una nueva herramienta

Pasaremos a explicar más detenidamente los pasos necesarios para llevar a cabo la implementación de una nueva herramienta en ETiles, o para mejorar una ya existente.

Lo primero que tenemos que hacer es tener claro que tipo de herramientas vamos a añadir a nuestro editor, y la funcionalidad que va a desempeñar.

Tomaremos como ejemplo una de las herramientas ya desarrolladas para hacer el seguimiento paso a paso, la herramienta Rellenar.

C.2.1. Paso 1: Creación de la función

El primer paso consiste introducir una función dentro de la clase Mapa dentro del fichero mapa.h.

```
void Rellenar(QRect rect);
```

A continuación definirla en la clase Mapa dentro del fichero mapa.cpp.

```
void Mapa::Rellenar(QRect rect)
{
    int i = listatiles->vistalistatiles->TileSeleccionado();
    //Si tenemos un tile seleccionado
    if(i!=-1)
    {
        int x= (int)(rect.x()/tanchuraZ);
        int y= (int)(rect.y()/talturaZ);
        if(posLibre(x,y))
        {
            matrizTiles[x][y]=i;

            minimapa->Actualizar(QPoint(x,y),
                                 pixmap,true);

            InsertarDeshacer(i,QPoint(x,y));
            InsertarNumDeshacer(0,1);
        }
    }
    update(rect);
}
```

Como podemos observar en el ejemplo, si la herramienta que hemos creado modifica en algún momento algún tile de nuestra matriz debemos repintar con la función update(rectangulo), cuyo rectángulo debe identificar la zona del mapa que ocupa el tile.

Además, si también queremos implementar que la acción que produce nuestra herramienta pueda ser deshecha y rehecha, tenemos que insertar en la lista deshacer el tipo de acción que se produce y el número de tiles que se modifican con dicha acción.

```
InsertarNumDeshacer(0,1);
```

Tenemos tres posibles acciones:

- Si es 0, indicamos que el una inserción.
- Si es 1, indicamos que es una eliminación.
- Si es 2, indicamos que se trata de un movimiento de posición.

Aparte, en otras dos listas almacenamos el tile al cual se produce el cambio y la posición que ocupa dentro de la matriz.

```
InsertarDeshacer(i,QPoint(x,y));
```

Si en cambio, nuestra herramienta produce otro tipo de acción aparte de las descritas, debemos modificar las funciones Deshacer y Rehacer. Ésto no conlleva ninguna dificultad, sólo tenemos añadir un nuevo *if* e implementar el cambio. Mostraremos el ejemplo de código cuando la acción es una inserción.

```
if(numtiles.x()==0) // 0 cuando se inserto algun tile
          {
              for(int i=0;i<numtiles.y();i++)
              {
                  //Tomamos el ultimo elemento y luego
                  //lo borramos de la lista deshacer.
                  int tile = tilesdeshacer.back();
                  QPoint pos = posdeshacer.back();
                  tilesdeshacer.pop_back();
                  posdeshacer.pop_back();

                  QRect rect(pos.x()*tanchuraZ,
                             pos.y()*talturaZ,
                             tanchuraZ,talturaZ);
                  //Borramos el tile de la matriz.
                  matrizTiles[pos.x()][pos.y()] = -1;

                  InsertarRehacer(tile,pos);

                  update(rect);
              }
          }
          else
              .......
```

Como se observa sólo cambiamos el valor de la matriz al valor vacío, y actualizamos la zona. También hay que tener en cuenta que hay que insertar el cambio en la lista de rehacer, que sigue la misma notación que deshacer.

En la medida de lo posible se deberá hacer comentarios en los aspectos más complicados del código, así como documentar la herramienta usando para ello la herramienta *Doxygen*.

C.2.2. Paso 2: Enlazar con los eventos del ratón

Puede ocurrir que nuestra herramienta necesite los eventos del ratón para llevarse a cabo, para ello disponemos de una serie de funciones que nos proporciona la librería Qt.

La función `mousePressEvent()`, se activa cuando se produce un evento de click sobre la superficie.

```
void Mapa::mousePressEvent(QMouseEvent *event)
 {
     QRect square = TileSeleccionado(event->pos());

     int x= (int)(event->pos().x()/tanchuraZ);
     int y= (int)(event->pos().y()/talturaZ);
     bool vacio = posLibre(x,y);

```

```
        if(pegado) //si tenemos accionada la accion pegar
        {
            ...
        }//si no esta pegado activo
        else if(borrar) // si borrar se encuentra activo
        {
            ...
        }
        else if(relleno) //si rellenar esta activo
        {
               Rellenar(square);
        }
        ...
}
```

Sólo añadimos un nuevo flujo a través de un *if* y de la variable controladora correspondiente, realizando la llamada a la función creada anteriormente.

Aquí aparece por primera vez alguna función desconocida, como puede ser `TileSeleccionado(posicion)`. Esta función devuelve un rectángulo que representa la superficie que ocupa un tile dentro del mapa. También encontramos la función `posLibre(x,y)`, que nos indica si la posición de la matriz que se le pasa como parámetro esta ocupada ya.

Por otro lado, la función `mouseMoveEvent()`, se activa cuando se produce un movimiento con el ratón. Tenemos que tener en cuenta el seguimiento del ratón explicado párrafos más abajo.

```
  void Mapa::mouseMoveEvent(QMouseEvent *event)
 {
     if(posCorrecta(event->pos()))
     {
         int x= (int)(event->pos().x()/tanchuraZ);
         int y= (int)(event->pos().y()/talturaZ);

         QRect rect = TileSeleccionado(event->pos());
         if(pegado)
         {
            ....
         }
         else if(relleno)
         {
             Rellenar(rect);
         }
            ...
     }
 }
```

El proceso seguido para su implementación es el mismo. Insertamos un nuevo *if* que llame a nuestra función. En esta función hay que tener en cuenta que debemos comprobar siempre si la posición donde se encuentra el cursor es correcta, ya que si el cursor sale de la superficie de la ventana se accederá a posiciones inexistentes de la matriz.

C.2.3. Paso 3: Crear la función controladora

Una vez que ya tenemos la implementación de nuestra función y deseamos que pueda activarse en la aplicación, tenemos que crear una función controladora que indique cuando está activa nuestra herramienta.

```
void Mapa::RellenoActivar(bool i)
{
    QRect rect = rectsel;
    rectsel = QRect();
    update(rect);

    setMouseTracking(!i);
    relleno=i;
}
```

Está es la función a la que se llamará desde la ventana principal cada vez que acciones la herramienta. Su implementación es fácil, sólo hay que cambiar el valor de variable controladora a **true** o **false**, dependiendo si se activa o desactiva.

En el caso de la función Rellenar, ésta hace uso de `setMouseTracking(!i)`. Esta función establece si el seguimiento de ratón está habilitado.

- Si el seguimiento del ratón está desactivado (por defecto), el mapa sólo recibe eventos del movimiento del ratón cuando se arrastra después de haber hecho un click.
- Si el seguimiento del ratón está activado, el mapa recibe eventos con el movimiento del ratón, es decir, no hace falta hacer presionar y luego arrastrar.

C.2.4. Paso 4: Enlazar las acciones en la ventana principal

Llegados a este punto hemos terminado con la clase **Mapa**. Ahora concentraremos nuestros esfuerzos en modificar la interfaz gráfica del editor para añadir los enlaces y los iconos para poder utilizar nuestra herramienta.

Si deseamos que nuestra herramienta se acceda a través de la barra de herramientas principal, lo primero que debemos hacer es conseguir un icono que represente con exactitud la función que desempeña. Luego en el constructor de la clase **VentanaPrincipal** creamos la acción y la añadimos de la siguiente manera:

```
relleno = new QAction(QIcon("Imagenes/32x32/pintar32.png")
                      ,tr("Relleno"),this);
connect(relleno,SIGNAL(triggered())
        ,this,SLOT(RellenoActivado()));
ui->mainToolBar->addAction(relleno);
```

La librería Qt nos proporciona una clase **QAction** que mediante un icono y un cadena de texto de ayuda permite dar las funciones básicas para los elementos de nuestra barra de herramientas.

Luego conectamos la acción con la función controladora que explicaremos a continuación.

```
 void VentanaPrincipal::RellenoActivado()
{
    if(inicio!=0){
        if(mapa->RellenoActivo())
        {
            mapa->RellenoActivar(false);
            relleno->setChecked(false);
        }
        else
        {
            if(mapa->BorrarActivado())
            {
                mapa->BorrarActivar(false);
                borrar->setChecked(false);
            }
            else if(mapa->SeleccionActiva())
            {
                mapa->SeleccionActivar(false);
                seleccion->setChecked(false);
            }
            mapa->RellenoActivar(true);
            relleno->setChecked(true);
        }
    }
}
```

Puede parecer que está función es innecesaria, pero no es el caso. Ya que en la barra de herramientas solo será posible tener activada una acción al mismo tiempo, por lo que en el momento en que el usuario haga click en alguna de ellas, si alguna otra herramientas está activa se desactiva inmediatamente y se elimina el resaltado.

Si en cambio, deseamos que nuestra herramienta tenga un enlace en el menú, lo podemos añadir de forma fácil. *Qt Creator* nos ofrece un potente entorno en el que modificar los menús de forma gráfica. Sólo tenemos que abrir el fichero ventanaprincipal.ui e insertar el nuevo menú.

Por último, establecemos el enlace con la función desarrollada.

```
void VentanaPrincipal::on_actionCopiar_activated()
{
    if(inicio!=0)
        mapa->Copiar();
}
```

Como se observa, comprobamos si existe un mapa en edición, en cuyo caso, realiza la llamada.

Si hemos seguido todos los pasos anteriores, será el momento de compilar nuestro editor y ya dispondremos de nuestra nueva herramienta.

C.3. Inserción de un nuevo algoritmo para desarrollar mapas aleatorios

Puede que implementemos un nuevo algoritmo de desarrollo para generar mapas aleatorios. Si nuestro deseo es que pueda usarse desde el editor, debemos implantarlo en la herramienta que lleva a cabo esta generación. Como sabemos, ETiles provee un sistema de generación de nuevos mapas fácilmente ampliable, sólo debemos seguir una serie de pasos.

Lo primero que tenemos que hacer es desarrollar nuestra función. Para ello, debemos desplazarnos a la clase **Ventana Principal**, declarar nuestra función y definirla. Ya que esta función sólo se accederá desde la propia clase se recomienda declararla en la parte privada para que no pueda ser accesible desde fuera. Esta función debe modificar cada posición de la matriz de datos con alguno de los tiles que existen en la lista tiles. Además debemos tener en cuenta las indicaciones del usuario para generar el mapa de forma adecuada.

Cuando terminemos la implementación de la función tenemos que agregarla a la función encargada de controlar su llamada.

```
 void VentanaPrincipal::CrearMapaAleatorio(double tanchura,
 double taltura, double tnumanchura, double tnumaltura,
QList<QPixmap>ltiles, QList<QString>lnombres, int aguatierra,
int contislas, int monvar, int semilla,int eleccion)
{
    CrearNuevoMapa(tanchura,taltura,tnumanchura,tnumaltura);
    CrearVentanaTiles();
    CrearMinimapa();

    //Si al menos tenemos un tile en la lista.
    if(ltiles.size() > 0)
    {
        //Insertamos los tiles en la lista tiles.
        for(int i=0;i<ltiles.size();i++)
        {
            tile->addPiece(ltiles.at(i),lnombres.at(i));
        }
        //Si la eleccion es el algoritmo Perlin Noise.
        if(eleccion==0)
        {
            ...
        }
        else
            ...
```

Como podemos observar se crean e inicializan todas las partes del editor, luego si se han añadido tiles comprobamos que algoritmo ha elegido el usuario con la variable `eleccion`. Por tanto, solo hay que añadir un nuevo flujo con la sentencia `if` que realice la llamada a nuestra función.

Para terminar tenemos que modificar el diálogo que permite introducir los datos para la generación del mapa. Este diálogo se implementa en la clase **dmaleatorios**. Accediendo al constructor incluimos una nueva línea que agregue el nuevo algoritmo a la lista de selección.

```
1        ui->algsel->addItem(QString(nombreDelAlgoritmo));
```

Con todo esto ya podemos usar nuestro nuevo algoritmo para generar un nuevo mapa aleatorio. Mencionar que si también queremos usar el algoritmo en la biblioteca deberemos realizar los mismo cambios explicados anteriormente en el fichero correspondiente.

Apéndice D

GNU Free Documentation License

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.
This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.
We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "**Document**", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "**you**". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.
A "**Modified Version**" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.
A "**Secondary Section**" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject.

(Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.
The "**Invariant Sections**" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.
The "**Cover Texts**" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.
A "**Transparent**" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "**Opaque**".
Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.
The "**Title Page**" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.
The "**publisher**" means any person or entity that distributes copies of the Document to the public.
A section "**Entitled XYZ**" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "**Acknowledgements**", "**Dedications**", "**Endorsements**", or "**History**".) To "**Preserve the Title**" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.
The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.
You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled “History”, Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled “History” in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the “History” section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled “Acknowledgements” or “Dedications”, Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled “Endorsements”. Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled “Endorsements” or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version’s license notice. These titles must be distinct from any other section titles.
You may add a section Entitled “Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.
You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.
The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.
In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.
You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.
If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.
If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.
However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.
Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.
Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.
Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.
"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.
"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.
An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.
The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright © YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with … Texts." line with this:

> with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.
If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Acrónimos

- BitMaP (BMP)
- Entorno de Desarrollo Integrado (IDE)
- GNU Image Manipulation Program (GIMP)
- GNU No es Unix (GNU)
- General Public License (GPL)
- Interfaz Gráfica de Usuario (GUI)
- Joint Photographic Experts Group (JPEG)
- Lesser General Public License (LGPL)
- Modelo Vista Controlador (MVC)
- PHP Hypertext Pre-processor (PHP)
- Portable Network Graphics (PNG)
- Simple DirectMedia Layer (SDL)
- eXtreme Programming (XP)

Bibliografía

[1] Agile Manifiesto. http://www.agilemanifesto.org/, 2001.

[2] A. Cockburn. *Agile Software Development , ISBN: 0-201-69969-9*. Addison-Wesley Professional, 2001. ISBN 0-201-69969-9.

[3] Cynthia Andres, Kent Beck. *Extreme Programming Explained: Embrace Change, ISBN: 0-321-27865-8*. Addison-Wesley Professional. 2ª edición, 2004. ISBN 0-321-27865-8.

[4] Aburruzaga García, Gerardo; Medina Bulo, Inmaculada y Palomo Lozano, Francisco. *Fundamentos de C++ , ISBN: 84-9828-007-9*. Servicio de Publicaciones de la Universidad de Cádiz. 2ª edición, 2006. ISBN 84-9828-007-9.

[5] Qt Reference Documentation. http://doc.qt.nokia.com/4.6/index.html, 2009/2010.

[6] Simple DirectMedia Layer - Official Site. http://www.libsdl.org/, Consultado: Junio 2010.

[7] David S. Ebert, F. Kenton Musgrave, Darwyn Peachey, Ken Perlin, Steven Worley. *Texturing & Modeling - A Procedural Approach, ISBN: 0-122-28730-4*. Morgan Kaufmann. 2ª edición, 1998.

[8] Carlos Villegas Núnez. forja: ETiles: Información del proyecto. https://forja.rediris.es/projects/editor-map-tile/.

[9] Bernardo Cascales Salinas. *El libro de LaTeX , ISBN: 84-205-3779-9*. Pearson Educación, 2003. ISBN 84-205-3779-9.

[10] IconFinder. http:http://www.iconfinder.com/, Consultado: Octubre 2010.

[11] Antonio García Alba. Tutorial wiki de libSDL para la programación de videojuegos. http://softwarelibre.uca.es/wikijuegos, 2008.

[12] G. J. Myers. *The art of software testing, ISBN: 0-471-46912-2*. Hoboken: John Wiley & Sons. 2ª edición, 2004. ISBN 0-471-46912-2.