Ingeniería Técnica en Informática de Gestión
TFC - Plataforma Gnu/Linux

Proyecto: Desarrollo de un videojuego para GNU/Linux
Documento: Memoria

- 13 de Junio de 2011 -

Autor: Antonio Martínez Martín
Profesor: Helena Rifá Pous
Consultor: Jordi Massaguer Pla

Índice

1. Plan de Trabajo....5
1.1. Introducción y objetivos....5
1.2. Descripción del Trabajo....5
1.2.1. Tareas....6
1.2.2. Hitos / Entregables....6
1.3. Metodología/Herramientas....6
1.3.1. Planificación temporal....7
2. Diseño del videojuego....8
2.1. Introducción....8
2.2. Alcance....8
2.3. Tipo de Videojuego....8
2.4. Audiencia....9
2.5. Requerimientos de Hardware....9
2.6. Argumento....9
2.7. Objetivos del juego....10
2.8. Lógica de juego....10
2.9. Controles....11
2.10. Interfaz....11
2.11. Aspecto Audiovisual....12
2.12. Ampliaciones....13
2.13. Distribución....14
3. Implementación....15
3.1. Entorno de desarrollo....15
3.1.1. Distribución GNU/Linux: Ubuntu....15
3.1.2. Lenguaje de programación: C....16
3.1.3. GCC (GNU Compiler Collection)....16
3.1.4. GNU Make....17
3.1.5. Editor de código fuente: gedit....18
3.1.6. GIMP (GNU Image Manipulation Program)....18
3.1.7. SDL (Simple Directmedia Layer)....19
3.1.7.1. Introducción....19
3.1.7.2. Subsistemas de SDL....19
3.1.7.2.1 Inicialización....19
3.1.7.2.2 Video....20
3.1.7.2.3 Gestión de ventanas....22
3.1.7.2.4 Gestión de eventos....23
3.1.7.2.5 Temporización....24
3.1.7.3 SDL_Mixer....25
3.1.7.4. Ventajas e inconvenientes de SDL....26
3.2. Diseño Técnico....27
3.2.1. Introducción....27
3.2.2. Mapa....27
3.2.2.1. Diseño del mapa....27
3.2.2.2. Desplazamiento....28

3.2.3. Personajes..... 29
3.2.3.1. Organización de los gráficos.....29
3.2.3.2. Animación..... 30
3.2.3.3. Comportamiento de los pandilleros.....31
3.2.3.4. Control del policía con el teclado..... 32
3.2.4. Detección de colisiones..... 33
3.2.5. Presentación y texto en pantalla..... 34
3.2.5.1. Marcadores durante la partida..... 34
3.2.5.2. Menú de inicio..... 35
3.2.5.3. Configuración de las teclas.....35
3.2.5.4. Mejores puntuaciones..... 35
3.2.5.5. Puntuación final.....35
3.2.6. Sonido..... 36
4. Código fuente..... 36
5. Conclusión.....36
6. Referencias.....37
Apendice A: Capturas de pantalla.....38

1. Plan de Trabajo

1.1. Introducción y objetivos

Actualmente los escritorios GNU/Linux apenas disponen de videojuegos de calidad desarrollados específicamente para ellos. La mayoría de las compañías profesionales centran su producción en Microsoft Windows y en diversas videoconsolas comercializadas por Nintendo, Sony y la propia Microsoft.

Gracias a Wine (http://www.winehq.org/about) podemos instalar y ejecutar muchos de los videojuegos más recientes para Windows natívamente en GNU/Linux, pero la situación no es óptima ya que a menudo surgen problemas de compatibilidad y para usuarios con poca experiencia puede resultar engorroso el manejo de este programa.

En los próximos años, con el aumento de la adopción de GNU/Linux en el escritorio y la proliferación de smartphones y otras plataformas portátiles con sistemas basados en GNU/Linux, la ventaja de Windows sobre Linux en el escritorio seguirá disminuyendo, y en el terreno de los videojuegos puede que muchas empresas empiecen a desarrollar juegos para GNU/Linux.

Algunos juegos de bajo presupuesto para Windows o Mac OS son versiones para escritorio de juegos desarrollados en principio para smartphones que, gracias al uso de APIs como SDL y OpenGL, son fácilmente portables. El objetivo del TFC es desarrollar un videojuego de este tipo para escritorios GNU/Linux.

1.2. Descripción del trabajo

El videojuego a desarrollar será el típico dotado de gráficos 2D en el que la acción transcurre en perspectiva top-down.

El jugador adoptará el papel de un agente de policía que acude a una zona conflictiva de la ciudad en respuesta a una llamada recibida en su comisaría.

Cuando llega al sitio se da cuenta de que no es bien recibido. El objetivo será evitar ser abatido por decenas de pandilleros que le saldrán a su paso por todas partes.

La puntuación obtenida se corresponderá con el número de pandilleros eliminados.

1.2.1. Tareas

1.Diseño del videojuego :

Especificación de la lógica de juego y diseño de su aspecto visual.

2.Estudio del API multimedia:

Familiarización con el API para desarrollo de aplicaciones multimedia que usaremos en la etapa de implementación.

3.**Implementación:**

Implementar la gestión de los recursos multimedia y a continuación la lógica del videojuego.

4.Pruebas:

Durante el transcurso del desarrollo se irán probando las distintas partes desarrolladas a medida que se vayan integrando, y una vez concluido se jugará repetídamente para comprobar que el funcionamiento del juego es el deseado.

1.2.2. Hitos / Entregables

1.Código fuente junto a los recursos multimedia utilizados.
2.Instrucciones de compilación e instalación.
3.Memoria
4.Presentación (diapositivas/video).

1.3. Metodología/Herramientas

Para la edición de los gráficos se usará GIMP, y para la implementación del diseño se empleará el lenguaje de programación C junto al API multimedia SDL (http://www.libsdl.org), que finálmente parece lo más adecuado a las necesidades en este caso, para obtener un buen rendimiento en la ejecución del código, y también por productividad, debido al escaso tiempo disponible, por su flexibilidad, su carácter multiplataforma y su facilidad de uso.

Otro punto a favor de SDL son las facilidades que nos ofrece para trabajar con el estándar profesional OpenGL(http://www.opengl.org) en nuestras aplicaciones, por si lo queremos utilizar para implementar algunas funcionalidades más complejas.

La metodología de desarrollo, tanto del código como de los gráficos, consistirá en ir implementando

, probando y mejorando iteratívamente aspectos y funcionalidades del diseño, para ir integrándolos en el código principal.

1.3.1. Planificación

La planificación estimada es la siguiente:

15 de Abril: Informe 1: Diseño del videojuego. (~15 páginas)

23 de Mayo: Informe 2: Implementación (~25 páginas)

13 de Junio: Memoria del proyecto (Plan de Trabajo + informe 1 + informe 2) y versión 1.0 del videojuego.

17 de Junio: Presentación (diapositivas/video)

2. Diseño del videojuego

2.1. Introducción

El diseño del videojuego se recoge en este apartado. Consiste en una descripción en líneas generales de las características básicas del videojuego a nivel conceptual y sirve como referencia entre todas las personas involucradas en el desarrollo.

Este apartado es un borrador que se irá completando durante todo el desarrollo a medida que se vayan fijando las carácteristicas del videojuego, es dinámico, y está sujeto a cambios y modificaciones contínuamente.

2.2. Alcance

El alcance del desarrollo viene limitado principálmente por el tiempo disponible (menos de 3 meses) y por las características del equipo, integrado por un solo miembro encargado de realizar todas las funciones: diseño, programación, gráficos, sonido, documentación, pruebas,...

Los videojuegos comerciales son programas informáticos complejos, e incluso aquellos con diseño y gráficos más sencillos destinados a jugadores ocasionales, cuentan con presupuestos considerables , con periodos más prolongados para las distintas fases de desarrollo, con equipos de personas bien nutridos y con editores de gráficos, de niveles, middleware avanzado de terceros y otras herramientas multimedia propias especializadas. Por lo cual, en este trabajo no se pretende completar un videojuego con un acabado profesional.

2.3. Tipo de Videojuego

El tipo de videojuego elegido se adapta bien al alcance del proyecto. Pertenece al género de los videojuegos de disparos en 2D o *shooter* en los que la acción transcurre en perspectiva *top-down*.

Se podría a su vez incluir en el grupo de los videojuegos *casuales*, con un diseño base que en principio no tiene demasiados elementos, aunque es fácilmente ampliable con detalles artísticos o con distintas modalidades de juego para aumentar el atractivo de cara a los jugadores.

Los juegos *casuales* tienen reglas de juego muy simples, un bajo nivel de estrategia y no requieren mucha práctica, habilidades especiales o largas sesiones de juego.

2.4. Audiencia

Usuarios de GNU/Linux, y usuarios ocasionales o habituales de juegos de acción que quieran partidas rápidas y cortas.

2.5. Requerimientos de Hardware

Serán estimados más precisamente al finalizar las pruebas del programa. En principio se espera que la instalación requiera poco espacio de disco duro y que el programa sea mínimamente eficiente para que no se necesite un ordenador con un procesador muy rápido o con mucha memoria R.A.M. en comparación con juegos más grandes.

Al no utilizar por ahora funcionalidades 3D podemos asegurarnos de que la ejecución del videojuego no será muy exigente con el hardware.

2.6. Argumento

En el tipo de videojuego elegido, en el que prima la acción, los argumentos no suelen ser muy elaborados, a diferencia de los argumentos que se pueden encontrar en otros tipos como juegos de rol, de estrategia, etc.

Para el argumento de este juego bastará con colocar al personaje protagonista en una situación en la que se produce un tiroteo.

El protagonista en este caso es un agente de policía de una ciudad peligrosa que, mientras está sentado en su comisaría , recibe una llamada de algún denunciante y se dirige al lugar de los hechos.

Al llegar, como si le estuvieran esperando y sin motivo aparente, decenas de pandilleros empiezan a salir por todas partes y se abalanzan sobre él rodeándole. Afortunádamente, fue el primero de su promoción en la academia de policía y sabe cómo actuar en estos casos. Al parecer, la llamada la hizo un bromista.

También podríamos darle al jugador la opción de elegir al principio de la partida entre varios protagonistas con distintos argumentos. Por ejemplo en la versión 2.0. , en lugar del policía podría elegir a un pandillero que sea miembro de otra pandilla y el argumento sería una pelea de pandillas.

2.7. Objetivos del juego

El principal objetivo del videojuego en el único modo de juego inicialmente disponible será aguantar vivo el mayor tiempo posible eliminando pandilleros.

La puntuación obtenida se podría computar en función del tiempo transcurrido desde el inicio hasta el fin de la partida y del número de pandilleros eliminados.

2.8. Lógica de juego

Los elementos del juego son entonces los siguientes:

Policía protagonista: El personaje que controla el jugador. Cuenta de inicio con su arma reglamentaria. Para aumentar la dificultad del juego se podría decidir durante la implementación si se limita el número de municiones, si puede recargar con una tecla, o si los pandilleros, al ser eliminados, dejan en el suelo cargadores. Si el jugador se quedase sin munición tendría que acudir rápidamente a recoger el cargador antes de que otros pandilleros le impidiesen el paso.

Pandilleros: Personajes a los que se tiene que enfrentar el policía. En principio podrían ir equipados con algún objeto contundente, o solo con los puños, y a medida que avanza la partida , para subir la dificultad, algunos pandilleros podrían llevar pistola.

También se podrían implementar niveles de agresividad o distintos comportamientos para los pandilleros. Por ejemplo, que intenten mantener rodeado al jugador, que a veces ataquen todos a la vez, que algunos corran más que otros, etc...

Terreno de juego: Tanto el policía como los pandilleros podrán hacer desplazamientos multidireccionales por el terreno de juego. El nivel de prueba en el modo de juego incluido será un extenso parque u otro sitio de la ciudad que abarque varias pantallas.

Al iniciar la partida, el jugador aparece situado en la parte central de la zona de juego y al efectuar los desplazamientos la *cámara* lo mantendrá siempre en el centro de la imagen.

El programa mantendrá un registro de la posición y del estado de los pandilleros a medida que vayan llegando o vayan siendo eliminados.

La partida acaba cuando los pandilleros le quitan al policía todos los puntos de salud. En ese momento se calculará la puntuación final y si el jugador consigue un nuevo record podrá introducir su nombre en la tabla de records.

2.9. Controles

Podrían ser configurables por el usuario a través del interfaz o mediante un fichero de configuración.

El control de la interfaz en la pantalla de inicio, menú de opciones y records se haría con los cursores o ratón para navegar y la tecla *enter* para aceptar.

Los controles ofrecidos por defecto al jugador durante una partida serían uno o dos de los siguientes grupos:

Solo teclado:	Con 4 teclas para desplazamiento en 8 direcciones (por ejemplo, a,s,d,w), 4 teclas para apuntar en 8 direcciones (j,k,l,i) y una tecla para disparar (espacio).
Teclado y ratón:	Con 4 teclas para desplazamiento en 8 direcciones(a,s,d,w), ratón para apuntar, y botón izquierdo del ratón para disparar.

2.10. Interfaz

Al ejecutar el programa, junto al nombre del videojuego, se mostrará un menú con varias opciones: Jugar, Controles, Instrucciones, Records, Créditos, Salir. Si pasados unos segundos no elegimos ninguna opción, se mostrará primero un fragmento de una partida grabada, después la tabla de records y a continuación se volverá a la pantalla de inicio. Presionando cualquier tecla también se volverá a la pantalla de inicio.

Jugar:	Seleccionando jugar iniciaremos una partida nueva. El interfaz durante la partida mostraría diversos marcadores de interés para el jugador como los puntos de salud, el nº de enemigos eliminados, el tiempo transcurrido, el arma seleccionada, la munición restante, etc. Se podría utilizar toda la pantalla para el juego y superponer en los márgenes los marcadores formados por cádenas de caracteres.
Controles:	Consultar controles por defecto o personalizarlos.
Instrucciones:	Muestra un resumen de las instrucciones del juego.
Records:	Muestra la tabla de records con las puntuaciones de otras partidas. Al concluir una partida se mostrará un dialogo para introducir el nombre del jugador consigue puntos suficientes.
Créditos:	Muestra los nombres y cargos del equipo de desarrollo (en este caso hay 1 miembro con varios cargos).
Salir:	Muestra un dialogo con las opciones Sí (terminar) | No (pantalla de inicio).

2.11. Aspecto Audiovisual

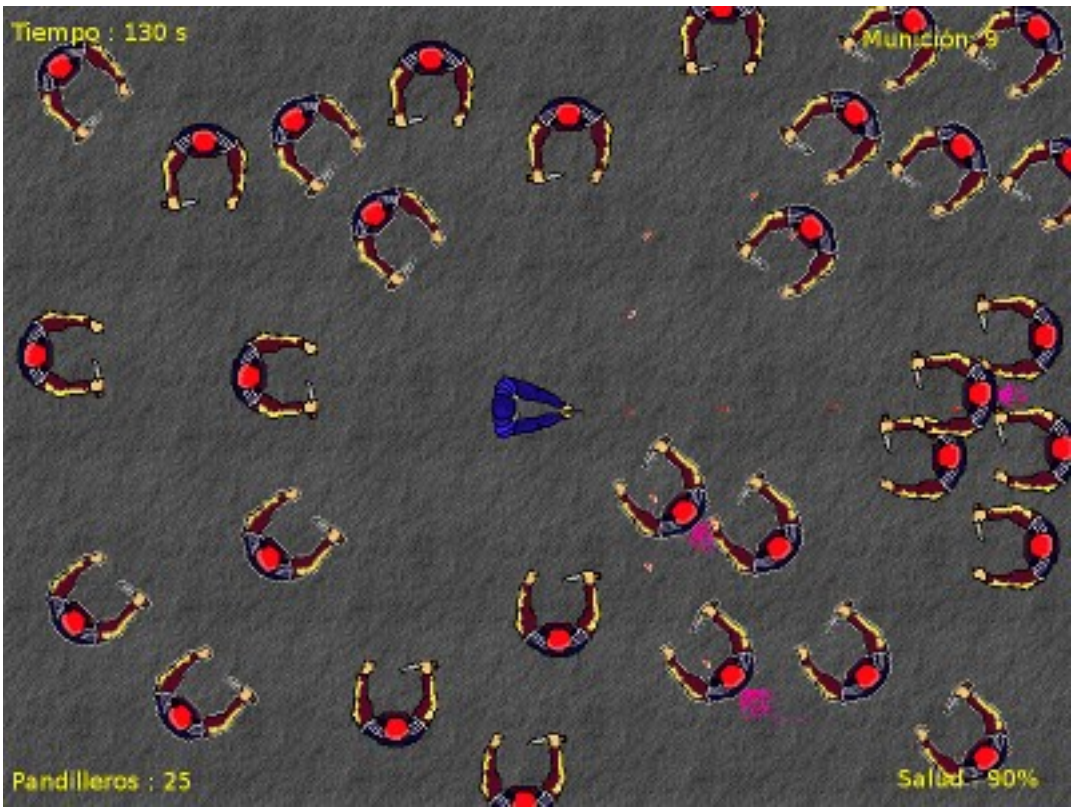
Para representar los elementos del juego se usarán diversos *bitmaps* elaborados con GIMP, con los gráficos representando en varias orientaciones al policía y a los pandilleros, con las que realizar las animaciones de sus movimientos.

El terreno de juego se dibujará a partir de varios pequeños *bitmaps* para aumentar la eficiencia del programa y para ahorrarnos trabajo artístico.

De inicio sería deseable no pasar demasiado tiempo con el trabajo gráfico realizando animaciones muy elaboradas ya que la calidad de estos/as no influirá demasiado en la jugabilidad básica, aunque el aspecto gráfico siempre le aporta personalidad a un juego.

Al policía se le puede representar siempre en posición de disparo en varias direcciones con 1 solo *bitmap* y para los pandilleros podemos tener 2 o 3 *bitmaps*, por ejemplo, uno que lo represente caminando con los brazos a la misma altura, y otro que lo represente atacando con un brazo encogido y otro extendido. Se restarían puntos de salud al policía si el pandillero le toca con el brazo extendido.

Para los impactos de bala en los pandilleros podríamos realizar una animación un poco más elaborada de sangre salpicando, ya que no requiere mucho tiempo y a los jugadores de videojuegos les gustan bastante estos detalles.



Para aumentar el realismo y darle más vida al juego se pueden usar algunos sonidos: de disparos, de impactos, gritos, pisadas (para compensar la falta de animaciones caminando)... Se añadirán una vez que tengamos completas las primeras versiones jugables.

2.12. Ampliaciones

La versión 1.0 podría mejorarse en varios puntos. Estos quedarían reflejados ya en versiones posteriores:

–Partidas en red o en el mismo ordenador con varios jugadores simultaneos en modo colaborativo (refuerzos).

–Elementos interactivos en la calle: edificios, coches aparcados o circulando que exploten con los disparos, escaparates para romper, etc...

–Más tipos de juego, niveles o misiones: por ejemplo, ir avanzando dentro de un edificio hasta llegar al jefe de la pandilla.

–Más tipos de enemigos con diferentes habilidades/comportamientos...

–Mejoras en los gráficos y animaciones.

–Optimizar y organizar mejor el código fuente, implementar gráficos 3D con manteniendo la jugabilidad 2D, etc...

–Posibilidad de elegir otros personajes protagonistas como miembros de otra pandilla, etc. y que en lugar de usar pistola como el poli use un palo, cuchillo , etc.

2.13. Distribución

Diversas webs como *Sourceforge* (**http://www.sourceforge.net**) ofrecen hospedaje gratuito.

Al finalizar el desarrollo, si el resultado es satisfactorio, se podría distribuir el código fuente del videojuego a través de Internet en una web como *Sourceforge* con una licencia apropiada. El paquete también incluiría los recursos multimedia utilizados (gráficos, audio) , e instrucciones de compilación e instalación.

Posteriormente si se desea se continuaría con el desarrollo de mejoras o de nuevas versiones, con las ampliaciones ya propuestas o con las contribuciones de nuevos colaboradores.

3. Implementación

3. Entorno de desarrollo

3.1.1. Distribución GNU/Linux: Ubuntu

Ubuntu proporciona una buena facilidad para instalar los paquetes de software que necesitamos desde su repositorio. Es un sistema operativo basado en la distribución Debian GNU/Linux que cuenta con un gran número de usuarios (aproximádamente el 50% de los escritorios Linux, más de 12 millones), distribuido como software libre y abierto. Linux Mint, que a su vez está basada en Ubuntu, es la segunda distribución más popular según Distro Watch.

Ubuntu ha sido diseñado principalmente para uso en el escritorio, aunque también existen versiones server y netbook.

La instalación de Ubuntu se realiza generalmente mediante su Live CD. El sistema operativo Ubuntu se puede ejecutar diréctamente desde el CD, permitiendo al usuario probar el sistema operativo para compatibilidad con su hardware y disponibilidad de drivers.

El CD también contiene el instalador Ubiquity, que puede guiar al usuario a través del proceso de instalación permanente. Las imágenes de CD de todas las versiones actuales y anteriores están disponibles para descarga en el sitio Web de Ubuntu.

Ubuntu divide su software en cuatro dominios para reflejar diferencias en las licencias y en el grado de soporte disponible. Algunas aplicaciones no soportadas reciben actualizaciones de los miembros de la comunidad, pero no de Canonical Ltd. (empresa que mantiene y desarrolla Ubuntu).

	Software Libre	Software No Libre
Soportado	Principal	Restringido
No Soportado	Universo	Multiverso

El software libre incluye solo software que cumple los requerimientos de licencia de Ubuntu, que se corresponden más o menos con los de Debian. Entre las excepciones para la categoría principal se incluye el firmware por su importancia, aunque Canonical no tenga acceso a su código fuente.

A excepción de SDL, que se ha descargado de su web, y compilado, el resto de software se ha obtenido del principal repositorio de Ubuntu. Otras herramientas como gedit y GIMP ya venían instaladas.

3.1.2. Lenguaje de programación: C

Los lenguajes C y C++ siguen siendo los preferidos por los desarrolladores profesionales de videojuegos de gama alta. Son lenguajes de propósito general, con una gran cantidad de compiladores disponibles que producen binarios muy eficientes, y se utilizan también para desarrollar todo tipo de software, sobre todo de sistemas, como por ejemplo el kernel Linux, y compiladores, librerias e intérpretes de otros lenguajes de programación, aunque también se usa en ocasiones para desarrollo web usando CGI como interfaz entre aplicación web, servidor y navegador.

Para el desarrollo de este videojuego, C , que ya lo conocía y es más reducido que C++, me pareció una buena opción, aunque hacía algún tiempo que no lo usaba.

C fue originálmente diseñado por Dennis Ritchie sobre el sistema operativo UNIX en un sistema DEC PDP-11. Posteriormente UNIX se reimplementó en C, junto al compilador y la mayoría de las demás aplicaciones del sistema.

La mayoría de las ideas de C se derivan del lenguaje BCPL desarrollado por Martin Richards. La influencia de BCPL en C proviene indiréctamente a través del lenguaje B, escrito por Ken Thompson en 1970 para el primer sistema UNIX en el DEC PDP-7.

BCPL y B son lenguajes sin tipos. En cambio C proporciona varios tipos de datos. Los tipos fundamentales son caracteres, enteros y numeros de punto flotante de varios tamaños. Además, hay una jerarquía de tipos derivados creados con punteros, arrays, estructuras y uniones, y los punteros proporcionan una aritmética de direcciónes independiente de la arquitectura. Con C es fácil implementar programas que se ejecuten en una gran variedad de hardware.

Durante muchos años, la definición de C era el manual de referencia en la primera edición de El Lenguaje de Programación C. En 1983, el Instituto Nacional de Estándares Americano (ANSI) estableció un comité para proporcionar una definición moderna y completa de C. La definición resultante, el estándar ANSI, fue completada a finales de 1988.

3.1.3. GCC (GNU Compiler Collection)

Es un repertorio de compiladores que distribuye la FSF (Free Software Foundation) bajo GNU GPL (General Public License) y que forma parte de la cadena de herramientas GNU (GNU's NOT UNIX) , el cual es un proyecto iniciado por Richard Stallman en 1983.

Originalmente GCC era el GNU C Compiler, ya que solo compilaba el lenguaje C, pero pronto se fueron añadiendo extensiones para compilar otros lenguajes como C++, Fortran, Pascal, Objective-C, Java, Ada...

Actualmente GCC es de uso estándar en la mayoría de sistemas tipo UNIX como Linux, la familia BSD y Mac OS X, se usa para desarrollar todo tipo de software profesional y está disponible para una gran variedad de arquitecturas de procesadores y plataformas. Solo código que use carácteristicas especificas de un sistema ha de ser modificado, el resto es fácilmente portable.

Para compilar con GCC el usuario invoca en la línea de ordenes el programa gcc, el cual interpreta los argumentos que le pasamos y decide que compiladores de lenguaje usar con cada fichero. La salida la pasa por su ensamblador y a continuación utiliza el enlazador para producir un binario ejecutable.

3.1.4. GNU Make

GNU Make es una herramienta que controla la generación de ejecutables y de otros ficheros no fuente, de un programa a partir de los ficheros fuente del program.

Make sabe como construir nuestro programa a partir de un fichero llamado Makefile, que lista cada uno de los ficheros no fuente y como computarlos a partir de los otros ficheros. Cuando escribimos un programa es útil escribir un Makefile para él, de forma que se pueda usar Make para construir e instalar el programa.

Make permite al usuario final construir e instalar nuestro programa sin que este tenga que conocer los detalles de como se hace, porque estos detalles vienen en el Makefile que suministramos.

Make decide automáticamente que ficheros necesita actualizar, basándose en que ficheros fuente han cambiado. También determina automáticamente el orden adecuado para actualizar los ficheros, en caso de que un fichero no fuente dependa de otro fichero no fuente.

Como resultado, si modificas unos cuantos ficheros fuente y después ejecutas Make, no se recompila todo el programa. Se actualizan solo aquellos ficheros no fuente que dependen directa o indiréctamente de los ficheros fuentes que has modificado.

Make también puede utilizarse para instalar o desinstalar un paquete, generar tablas de etiquetas para el, o cualquier otra cosa que se quiera hacer frecuentemente.

Las *reglas* en el Makefile dicen a Make como ejecutar una serie de comandos para construir un fichero *objetivo* a partir de ficheros fuentes. También especifica una lista de *dependencias* del fichero objetivo. Esta lista debería incluir todos los ficheros (fuente o otros objetivos) que se usan como entrada a los comandos en la regla.

Cuando ejecutamos Make, podemos especificar objetivos particulares; si no lo hacemos, Make actualiza el primer objetivo listado en el Makefile. Cualquier otro fichero objetivo requerido como entrada para generar estos objetivos será actualizado primero.

3.1.5. Editor de código fuente: gedit

Para editar el código fuente tenemos gedit, que es un editor de textos de propósito general y el editor de textos por defecto para escritorios GNOME, también disponible para MAC OS X y Windows.

Su interfaz gráfico es sencillo y ligero al igual que el entorno GNOME. Gedit usa GTK+ y las librerias GNOME. La integración con GNOME permite arrastrar y soltar sobre Nautilus, el gestor de ficheros de GNOME.

gedit incluye facilidades para la edición de código fuente como el resaltado de la sintaxis y comprobación de la ortografía del lenguaje, pestañas para editar varios ficheros de código fuente al mismo tiempo, opciones hacer/deshacer, busqueda y reemplazo de texto, números de linea, emparejamiento de llaves, resaltado de la linea actual, indentación automática, cambio de linea automático ,grabación automática de ficheros...

3.1.6. GIMP (GNU Image Manipulation Program)

GIMP es un programa de retoque fotográfico y de composición y creación de imágenes. Se puede ampliar mediante plug-ins y extensiones y mediante un interfaz de scripting avanzado que permite automatizar procedimientos complejos de manipulación de imágenes.

El desarrollo de GIMP se hace bajo X11 en plataformas UNIX, para las cuales se desarrollaron las primeras versiones a finales de los 90, aunque actualmente a partir del mismo código con pocos cambios tambień tenemos versiones para Windows y MAC OS X.

Desde su primera versión GIMP ha usado tres API de desarrollo de GUI diferentes. La primera versión utilizaba Motif, que dejó de usarse para desarrollar un API personalizado: el GIMP ToolKit (GTK) con el que reemplazar a Motif, y finálmente GTK fue reimplementado con orientación a objetos y cambió de nombre a GTK+.

Un aspecto un poco lioso para el usuario que no está acostumbrado a GIMP, es que el interfaz utiliza múltiples ventanas para contener las funcionalidades y cada imagen se abre en una ventana aparte. Se espera que esto cambie para la versión 2.8 con un rediseño del interfaz que de la opción de elegir un modo de ventana única. El modo de ventana única vendrá por defecto.

En el juego se han utilizado ficheros BMP para almacenar los gráficos que pasamos a SDL. GIMP nos permite trabajar con este formato. Las funcionalidades utilizadas son las básicas del lápiz, cubo de pintura, rotar imagen, aumentar/disminuir tamaño, selección, cortado, copiado, pegado, selector de colores...

3.1.7. SDL (Simple Directmedia Layer)

3.1.7.1. Introducción

SDL es un API de propósito general que proporciona acceso a bajo nivel a funcionalidades de audio, teclado, ratón, joystick, hardware 3D via OpenGL y un framebuffer 2D a través de múltiples plataformas.

La versión de SDL con la que se ha trabajado es la 1.2.14, que es la última versión estable a fecha de hoy, con los drivers para Linux de video y audio usados por defecto (X11 y dsp).

SDL está escrito en C, pero no hay problema en usarlo con una gran cantidad de lenguajes.

3.1.7.2. Subsistemas de SDL

En la carpeta **/SDL-1.2.14/docs** tenemos la documentación de SDL en formato HTML.

3.1.7.2.1 Inicialización

SDL se compone de 8 subsistemas - Audio, CDROM, Gestión de Eventos, E/S de Ficheros, Gestión de Joystick, Hilos, Temporizadores y Video. Antes de usar cualquiera de estos subsistemas deben ser inicializados llamando SDL_Init (o SDL_InitSubsystem). SDL_Init debe ser llamado antes de cualquier otra función SDL.

Automáticamente inicializa los subsistemas de Gestión de Eventos, E/S de ficheros e Hilos y acepta un parámetro especificando qué otro subsistema inicializar.

Ej. SDL_Init(SDL_INIT_VIDEO | SDL_INIT_TIMER) inicializa los subsistemas por defecto, el subsistema de video y el subsistema de temporización.

SDL_Init se complementa con SDL_Quit (y SDL_QuitSubSystem). SDL_Quit cierra todos los subsistemas, incluidos los subsistemas por defecto. Se debe llamar siempre antes de salir de una aplicación.

Los subsistemas de SDL a los que más he recurrido en la programación del videojuego con sus principales servicios son el subsistema de video, de gestión de eventos y de temporización. Y como andaba escaso de tiempo en lugar de utilizar el subsistema de audio de SDL he utilizado diréctamente el add-on SDL_Mixer, que nos gestiona el tema igual de cómodo que SDL en los otros subsistemas.

En la documentación podemos echar un vistazo rápido a todo lo que ofrece SDL en estos subsistemas. A continuación repaso algunas de las funciones de estos servicios que más he utilizado o que utilizaré en versiones posteriores para añadir algo al juego:

3.1.7.2.2 Video

SDL proporciona un interfaz muy simple para el framebuffer del display. El framebuffer se representa como una superficie fuera de pantalla sobre la que se puede escribir diréctamente. Si se quiere que la pantalla muestre lo que se hemos escrito, una llamada a la función *update* garantizará que la porción de pantalla especificada es actualizada.

Después de inicializar la librería, podemos arrancar el display de video de diferentes formas. La forma más fácil es elegir una resolución y una cantidad de colores habitual e inicializar el video, comprobando errores. Normálmente SDL nos devuelve el modo solicitado, pero en otras ocasiones SDL puede estar emulando y convirtiendo el display al actualizarlo.

Lo mejor es preguntar a SDL por el modo de video que más se aproxime al deseado y a continuación convertir nuestras imágenes a su formato de pixel.

SDL soporta actualmente cualquier profundidad de bits mayor o igual a 8 bits por pixel. Los formatos 8 bpp se consideran modos 8-bit paletizados, mientras que los modos con 12, 15, 16, 24 y 32 bpp se consideran modos de “pixel empaquetado”, significando que cada pixel contiene los componentes RGB del color empaquetados en los bits del pixel.

Tras inicializar el modo de video, podemos tomar la superficie devuelta, y escribir sobre ella como sobre cualquier otro framebuffer, llamando a la rutina de actualización sobre la marcha.

Funciones y tipos del subsistema de video	
SDL_GetVideoSurface	Devuelve un puntero a la superficie de display actual.
SDL_GetVideoInfo	Devuelve un puntero a información sobre el hardware de video.
SDL_VideoDriverName	Obtiene el nombre del driver de video.
SDL_ListModes	Devuelve un puntero a un array de dimensiones de pantalla disponibles para el formato y banderas de video especificadas.
SDL_VideoModeOK	Comprueba que un modo de video particular es soportado.
SDL_SetVideoMode	Establece un modo de video con la anchura, altura y bits por pixel especificados.
SDL_UpdateRect	Actualiza el area especificada en una determinada pantalla.
SDL_Flip	Intercambia buffers de pantalla.
SDL_SetColors	Establece una parte del colormap para la superficie de 8 bits especificada.

SDL_SetPalette	Establece los colores de la paleta de una superficie de 8 bits.
SDL_UpdateRects	Actualiza una lista de rectangulos en pantalla.
SDL_SetGamma	Establece la función del gamma del color para el display.
SDL_GetGammaRamp	Obtiene las tablas de consulta del gamma del color para el display.
SDL_SetGammaRamp	Establece las tablas de consulta del gamma del color para el display.
SDL_MapRGB	Establece una correspondencia entre un valor de color RGB y un formato de pixel.
SDL_MapRGBA	Establece una correspondencia entre un valor de color RGBA y un formato de pixel.
SDL_GetRGB	Obtiene los valores RGB de un pixel en el formato de pixel especificado.
SDL_GetRGBA	Obtiene los valores RGBA de un pixel en el formato de pixel especificado.
SDL_CreateRGBSurface	Crea una superficie SDL vacia
SDL_CreateRGBSurfaceFrom	Crea una superficie a partir de información de pixeles.
SDL_FreeSurface	Libera (borra) una superficie SDL.
SDL_LockSurface	Cierra una superficie para acceso directo.
SDL_UnlockSurface	Abre una superficie cerrada préviamente.
SDL_LoadBMP	Carga un fichero Windows BMP en una SDL_Surface
SDL_SaveBMP	Salva una SDL_Surface como fichero Windows BMP.
SDL_SetColorKey	Establece la clave de color (pixel transparente) y aceleración RLE en una superficie *bliteable*.
SDL_SetAlpha	Ajusta las propiedades alfa de una superficie.
SDL_SetClipRect	Establece el rectángulo de *clipping* para una superficie.
SDL_GetClipRect	Obtiene el rectángulo de *clipping* de una superficie.
SDL_ConvertSurface	Convierte una superficie al mismo formato que otra superficie.
SDL_BlitSurface	Realiza una copia rápida (*blit*) de una superficie origen a una superfice destino.

SDL_FillRect	Realiza un relleno rápido del rectángulo espeficicado con un color.
SDL_DisplayFormat	Convierte una superficie al formato del display
SDL_DisplayFormatAlpha	Convierte una superficie al formato del display.
SDL_WarpMouse	Establece la posición del cursor del ratón
SDL_CreateCursor	Crea un nuevo cursor de ratón
SDL_FreeCursor	Libera un cursor creado con SDL_CreateCursor
SDL_SetCursor	Establece el cursor de ratón actuálmente activo.
SDL_GetCursor	Obtiene el cursor de ratón actuálmente activo.
SDL_ShowCursor	Conmuta mostrar o no mostrar cursor en pantalla.
SDL_Rect	Define un area rectángular
SDL_Color	Descripción de color independiente del formato.
SDL_Palette	Paleta de color para formatos 8 bpp
SDL_PixelFormat	Almacena información sobre el formato del la superficie.
SDL_Surface	Estructura de una superficie gráfica.
SDL_VideoInfo	Información sobre el sistema de video objetivo.

3.1.7.2.3 Gestión de ventanas

SDL proporciona un pequeño conjunto de funciones de gestión de ventanas que permiten a las aplicaciones cambiar su título y conmutar modo ventana y pantalla completa.

Funciones del subsistema de gestión de ventanas	
SDL_WM_SetCaption	Establece título de ventana y nombre de icono
SDL_WM_GetCaption	Obtiene título de ventana y nombre de icono
SDL_WM_SetIcon	Establece icono para ventana de display
SDL_WM_IconifyWindow	Iconifica/minimiza la ventana
SDL_WM_ToggleFullScreen	Conmuta modo pantalla completa
SDL_WM_GrabInput	Atrapa entrada de ratón y teclado.

3.1.7.2.4 Gestión de eventos

La gestión de eventos permite a la aplicación recibir entrada del usuario. La gestion de eventos se inicializa junto al video con la llamada SDL_Init(SDL_INIT_VIDEO);

Intérnamente, SDL almacena todos los eventos que esperan ser gestionados en una cola de eventos. Usando funciones como SDL_PollEvent y SDL_PeepEvents se pueden observar y gestionar los eventos de entrada pendientes.

La clave para la gestión de eventos en SDL es la unión SDL_Event. La propia cola de eventos está compuesta de una serie de uniones SDL_Event, una para cada evento pendiente. Las uniones SDL_Event se leen de la cola con la función SDL_PollEvent y a partir de ahí la aplicación se encarga de procesar la información almacenada en ellas.

Funciones del subsistema de gestión de eventos	
SDL_PumpEvents	"infla" el bucle de eventos, recogiendo eventos de los dispositivos de entrada
SDL_PeepEvents	Consulta si hay mensajes en la cola de eventos y opcionalmente los devuelve.
SDL_PollEvent	Pregunta por eventos actuálmente pendientes.
SDL_WaitEvent	Espera indefinidamente al próximo evento.
SDL_PushEvent	Coloca un evento en la cola de eventos
SDL_SetEventFilter	Establece un filtro para procesar todos los eventos antes de que sean posteados a la cola de eventos.
SDL_GetEventFilter	Obtiene un puntero al filtro de eventos.
SDL_EventState	Permite establecer el estado de procesar ciertos eventos.
SDL_GetKeyState	Obtiene una captura del estado actual del teclado.
SDL_GetKeyName	Obtiene el nombre de un keysim virtual de SDL.
SDL_EnableKeyRepeat	Establece la tasa de repetición del teclado
SDL_GetMouseState	Obtiene el estado actual del ratón.
SDL_GetRelativeMouseState	Obtiene el estado actual del ratón.
SDL_GetAppState	Obtiene el estado de la aplicación.

Estructuras del subsistema de gestión de eventos	
SDL_Event	Estructura general de eventos
SDL_ActiveEvent	Estructura de eventos de visibilidad de la aplicación.
SDL_KeyboardEvent	Estructura de eventos del teclado.
SDL_ResizeEvent	Estructura de evento de cambio de tamaño de ventana.
SDL_ExposeEvent	Quitar el evento solicitado.
SDL_SysWMEvent	Evento de gestor de ventana dependiente de plataforma
SDL_UserEvent	Tipo de evento definido por el usuario
SDL_QuitEvent	Quitar el evento pedido
SDL_Keysym	Estructura keysim
SDLKey	Definiciones keysim

3.1.7.2.5 Temporización

SDL proporciona varias funciones multiplataforma para gestionar el tiempo. Proporciona un modo de obtener el tiempo actual, un modo de esperar un poco y un mecanismo simple de temporización. Estas funciones nos proporcionan dos formas de mover un objeto cada x milisegundos:

1. Usando una función callback de temporización. Esta puede tener el efecto negativo de ejecutarse en un hilo aparte o de usar señales de alarma, pero es más fácil de implementar.

2. Tomando el número de milisegundos transcurridos, y mover el objeto si, por ejemplo, han pasado 30 milisegundos.

En el juego utilizo sobre todo el segundo método para mover los personajes, controlar sus animaciones, etc.

Funciones del subsistema de temporización	
SDL_GetTicks	Obtiene el número de milisegundos desde la inicialización de la librería SDL
SDL_Delay	Espera el número de milisegundos especificados antes de volver.
SDL_AddTimer	Añade un temporizador que llamará a un *callback* al transcurrir el número de milisegundos especificado.
SDL_RemoveTimer	Elimina un temporizador añadido con SDL_AddTimer.
SDL_SetTimer	Establece un callback que se ejecute después de que transcurra un número especificado de ms.

3.1.7.3 SDL_Mixer

SDL_Mixer es una librería add-on de mezcla de sonido usada junto a SDL y que es casi tan portable como esta. Permite al programador utilizar múltiples *samples* de sonido junto a música sin tener que codificar un algoritmo de mezclas. También simplifica la manipulación, carga y reproducción de muestras y músicas de todo tipo de formatos de audio.

SDL_Mixer se define en su README como un simple mezclador de audio multicanal que soporta 8 canales de audio de 16 bit stereo, más un solo canal de música, mezclado por el popular MikMod MOD, Timidity MIDI y las librerías MP3 SMPEG.

El mezclador puede cargar actualmente ficheros Microsoft WAVE y ficheros VOC de Creative Labs como muestras de audio, puede cargar ficheros MIDI mediante Timidity y los siguientes formatos de música mediante MikMod: .MOD .S3M .IT .XM.

Puede cargar *streams* Ogg Vorbis como música si se compila con las librerías de Ogg Vorbis, y finalmente, puede cargar música MP3 usando la librería SMPEG.

Al igual que SDL, SDL_Mixer también está disponible bajo licencia LGPL y para las necesidades de sonido del juego ha bastado con utilizar algunas de sus funciones de *samples* y canales con nuestros ficheros .WAV para conseguir el ambiente deseado en el juego. Habiendo dejado el sonido para el final ha habido suerte en que estos aspectos de SDL_Mixer erán bastante cómodos de utilizar.

Funciones de SDL_Mixer	
Mix_Init	Inicializa SDL_Mixer
Mix_Quit	Cierra SDL_Mixer
Mix_OpenAudio	Abre el mezclador de sonido
Mix_CloseAudio	Cierra el mezclador de sonido
Mix_LoadWAV	Carga un fichero con formato .WAV
Mix_VolumeChunk	Establece el volumen de la mezcla.
Mix_FreeChunk	Libera un *sample*
Mix_AllocateChannels	Establece el número de canales a mezclar
Mix_Volume	Establece el volumen de mezcla de un canal
Mix_PlayChannel	Toca un bucle
Mix_HaltChannel	Para la reproducción en un canal
Mix_Playing	Obtiene el estado de reproducción de un canal
Mix_GetChunk	Obtiene que *sample* se reproduce en un canal.

3.1.7.4. Ventajas e inconvenientes de SDL

Algunos puntos a favor o ventajas de SDL:

- Es software libre y de código abierto.
- Portabilidad. Facilita el desarrollo de software para una gran variedad de sistemas.
- Facilidad de uso frente a otras APIs como DirectX (en Windows) o X11, sobre las que SDL se apoya.
- Estabilidad. Se le encuentran pocos errores.
- Proporciona diversos subsistemas, no solo video, lo cual es una ventaja en Linux frente a usar solo otro API como OpenGL.
- Facilita la configuración y uso de OpenGL
- Flexibilidad. Está diseñada para proporcionar un mínimo útil para todo tipo de aplicaciones multimedia, no solo videojuegos.

Algunos puntos en contra o inconvenientes de SDL:

- Actualmente no ofrece aceleración por hardware, aunque si lo hará a partir de la versión 1.3. Se puede solucionar utilizando OpenGL junto a SDL, si queremos desarrollar aplicaciones más complejas o que manejen resoluciones de pantalla mayores.
- No ofrece de entrada algunas funciones que podrían ser útiles para el manejo de más formatos gráficos, de texto en pantalla, etc..., aunque esto también se puede considerar una ventaja para no perdernos entre miles de funciones. De todas formas hay add-ons disponibles o si lo preferimos podemos implementar nosotros la funcionalidad requerida.

3.2. Diseño Técnico

3.2.1. Introducción

Para el diseño técnico he adaptado el aspecto gráfico de una resolución de 800x600 con 24 o 32 bits/pixel a otra de 320x240 con 16 bits/pixel:

- Requiere menos detalle y menos tiempo para su elaboración. Más tiempo para animaciones.

- Aumenta la velocidad de ejecución: SDL no utiliza de momento aceleración por hardware (habría que utilizar OpenGL o ver si X11 está usando OpenGL) y prefiero asegurar fluidez si alguien usa Linux en algún ordenador con bajo rendimiento, ya que no implementaré una opción para elegir entre varias resoluciones.

- Se jugará en una ventana o a pantalla completa.

3.2.2. Mapa

El terreno de juego está compuesto de la manera que suele ser habitual en este tipo de juegos, a partir de unas pocas *losas* de 32x32 pixels que se repiten para crear un escenario de gran tamaño por el que pueda moverse el protagonista. La ventaja de esta técnica es que tenemos un terreno de juego grande sin tener que hacer mucho trabajo artístico y al mismo tiempo se aligera el peso del paquete para su distribución. El mapa tiene un tamaño de 200x150 *losas.*

3.2.2.1. Diseño del mapa

Para distribuir las *losas* por el mapa se suele utilizar un editor de niveles cuando disponemos de una mayor variedad de *losas*. Como para este juego solo cuento con dos *losas* diferentes utilizo un par de bucles para distribuirlas:

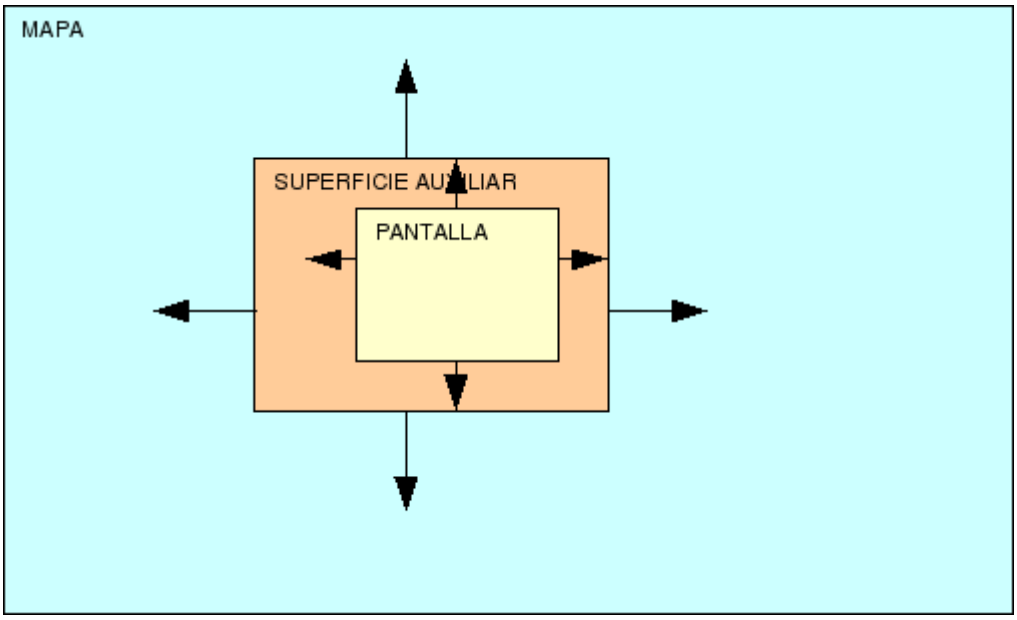
```
45 void loadTileMap(void)
46 {
47         int x,y;
48         int c=0;
49
50         for (y=0; y<TILEMAPH; y++)
51         for (x=0; x<TILEMAPW; x++)
52         {
53                 if((y%2)==0)
54                 {
55                         if((x%2)==0) tilemap[x][y]=0;
56                         else tilemap[x][y]=1;
57                 }
58                 if((y%2)!=0)
59                 {
60                         if((x%2)==0) tilemap[x][y]=1;
61                         else tilemap[x][y]=0;
62                 }
63         }
64 }
```

3.2.2.2. Desplazamiento

Con SDL es bastante sencillo implementar el desplazamiento del mapa para que el protagonista del juego se pueda mover en cualquier dirección.

Como disponemos de una gran cantidad de R.A.M. no habría problema en cargar el mapa completo, que ocuparía unos 1.8 MB. O de otra forma, como solo tenemos 2 *losas* diferentes ni siquiera habría que gestionarlo.

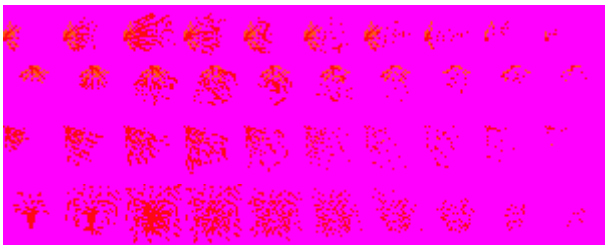
En la situación intemedia es mejor usar la memoria moderádamente si vamos a elaborar después mapas más extensos y también para poder usar más adelante más variedad de losas con un editor. Para ello gestionamos una superficie auxiliar que representa una porción del mapa, la cual vamos actualizando a medida que sea necesario para reflejar en pantalla la posición del personaje.



3.2.3. Personajes

3.2.3.1. Organización de los gráficos

He agrupado los gráficos en distintos ficheros .bmp con las animaciones de los personajes y los salpicones de sangre en algunas direcciones. Volteándolas horizontálmente y verticálmente obtenemos cómodamente el resto de las animaciones para las otras direcciones.



Al iniciar el programa los cargamos en las siguientes estructuras:

```
38      struct Animation_struct
39      {
40              SDL_Surface *frames[MAXFRAMES];
41              Uint8 nframes;
42              Uint32 framet;
43      };
44
45      typedef struct Animation_struct Animation;
46
47      struct Sprite_struct
48      {
49              char name[10];
50              Animation animations[NANIM];
51      };
52      typedef  struct Sprite_struct Sprite;
53
54
55
56      extern Sprite sprites[NSPRITES];
```

3.2.3.2. Animación

La animación de cada personaje la controlamos con los atributos charspr, curranim y currfr del tipo Char, que mantienen el gráfico, la animación y el fotograma que estamos utilizando:

```
29      struct Char_struct
30      {
31          Uint8 charspr;
32          Uint8 curranim;
33          Uint8 currfr;
34          Uint8 v;
35          Uint8 d;
36          Sint16 x,y;
37          Uint8 active;
38          int health;
39          Uint32 movet;
40          Uint32 movedt;
41          Spatter spatters[8];
42
43
44      };
45
46      typedef struct Char_struct Char;
```

En esta estructura mantenemos además otros atributos del personaje, como su posición en el mapa (x,y) el número de pixels que avanza en cada actualización de la posición (v), su dirección (d), si se encuentra activo (active), su salud (health), la velocidad de su carrera(movet,movedt) y los salpicones provocados por los impactos de las armas.

3.2.3.3. Comportamiento de los pandilleros

El comportamiento de los pandilleros puede ser tan complejo como queramos y es bastante mejorable.

Por ahora , para que el juego resulte ameno y variado, nos basta con que realicen un par de maniobras para cumplir con lo que se deseaba en este apartado en la etapa de diseño:

1. Persecución: El pandillero va directamente hacia el policia y le intenta atacar. Cada pandillero corre a una velocidad ligéramente diferente.

2. Paseo: El pandillero rompe filas y se da una vuelta en direcciones aleatorias por el mapa durante un tiempo determinado. A continuación vuelve al estado de persecución. Así el jugador tiene la sensación de que los pandilleros se mueven para intentar rodearle después y que no atacan todos a la vez. La aparición de nuevos pandilleros que atacan por la espalda al eliminar a alguno de sus compañeros también sorprende al jugador.

   Este aspecto se podría mejorar en la versión 0.2 para que los pandilleros realmente se distribuyan por el mapa intentando rodear al policia, esperando su turno para atacar.

```
120 void moveMuggers(void)
121 {
122         int i = 0;
123
124         while(i < MAXMUGGERS)
125         {
126                 if(SDL_GetTicks() - muggers[i].movedt > muggers[i].movet)
127                 {
128                         if (stepon(i)==1)
129                         {
130                                 states[i].state = WALK;
131                                 states[i].time = SDL_GetTicks();
132
133                         }
134                         if(states[i].state == WALK && (SDL_GetTicks()-states[i].time\
135                          < WALKT)) walk(i);
136                         else states[i].state = CHASE;
137                         if(states[i].state == CHASE) chase(i);
138                         chkDist(i);
139
140                         muggers[i].movedt = SDL_GetTicks();
141                 }
142                 i++;
143         }
144
145 }
```

Otra carácteristica necesaria del comportamiento de los pandilleros que se ha implementado es que después de pinchar al policia le den un tiempo de respiro para que se pueda mover a una distancia segura y seguir disparando.

```
378 void attack(int i)
379 {
380         muggers[i].curranim = muggers[i].d;
381         muggers[i].currfr = 0;
382
383
384
385         if(SDL_GetTicks() - stabt[i] >= STABT || stabt[i]==0)
386         {
387                 cop.health -= STABD;
388                 cop.spatters[selSpatter(muggers[i].curranim)].n++;
389                 stabt[i] = SDL_GetTicks();
390         }
391
392 }
```

Para todo esto el subsistema de temporización de SDL es de gran utilidad.

3.2.3.4. Control del policía con el teclado

La funciones moveCop( ) y aiming( ) en cop.c controlan el desplazamiento del policia y comprueban si está apuntando con la pistola respectívamente. Si está apuntando entonces puede disparar. El policía puede moverse, apuntar y disparar simultáneamente.

Una de las comprobaciones que realizamos en moveCop( ) para que la cámara acompañe al policia:

```
64          if ( keystates[MOVEDOWN] && keystates[MOVERIGHT] && \
65                  !keystates[MOVELEFT] && !keystates[MOVEUP] )
66          {       if(cop.x > 80) scrollRight();
67                  if(cop.y > 140) scrollDown();
68                  if(cop.x < MAXCOPX) cop.x += cop.v;
69                  if(cop.y < MAXCOPY) cop.y += cop.v;
70                  cop.d = SE;
71                  return;
72          }
```

3.2.4. Detección de colisiones

La detección de los impactos de navaja la hago por proximidad de cada pandillero con el policia. Cuando estos se acercan a una distancia corta extienden el brazo para pincharle.

```
353 void chkDist(int i)
354 {
355         Sint16 dx,dy,ccopx,ccopy,cmux,cmuy;
356
357         ccopx = cop.x + SPRITEFRAMEW/2;
358         ccopy = cop.y + SPRITEFRAMEH/2;
359         cmux = muggers[i].x + SPRITEFRAMEW/2;
360         cmuy = muggers[i].y + SPRITEFRAMEH/2;
361
362         dx = ccopx - cmux;
363         dy = ccopy - cmuy;
364
365         if (fabs(dx)<20 && fabs(dy)<20)
366         attack(i);
367 }
```

Para detectar los impactos de bala en los pandilleros comprobamos primero que la linea de disparo corta una circunferencia con centro en el pandillero, y después determinamos qué pandillero es el afectado calculando sus distancias al punto de disparo.

```
if(muggers[i].active)
{
        ccx = muggers[i].x + SPRITEFRAMEW/2;
        ccy = muggers[i].y + SPRITEFRAMEH/2;
        if(ptarget(ccx,ccy))
        {
                mugbulletd = dpl(ccx,ccy,firex,firey);
                if(mugbulletd <= IMPDIST)
                {
                   copmugd = dpp(ccx,ccy,firex,firey);
                   if(copmugd > -1)
                   {
                        if(copmugd < copmugmind)
                        {
                                hit = i;
                                copmugmind = copmugd;
                        }
                   }
                }
        }
}
```

Para garantizar la velocidad del cálculo de distancias he preparado las raices que voy a necesitar en un array. Esto limita el alcance máximo del disparo y gasta un poco de memoria, pero es rápido y fácil de usar.

En algunos casos como en el de los impactos de navaja, para comprobar si recoge un cargador, o si 2 pandilleros se pisan no haría falta usar estas funciones, pero las incluyo por claridad y por si aumenta el número de pandilleros activos.

3.2.5. Presentación y texto en pantalla

3.2.5.1. Marcadores durante la partida

Los marcadores utilizados durante la partida están compuestos por caracteres 8x8.

Para utilizarlos los cargamos desde sus ficheros .bmp en superficies SDL y a continuación solo tenemos que utilizar la superficie que guarda el gráfico del carácter que necesitamos para imprimir las cadenas de los marcadores en sus posiciones correspondientes:

```
while( s[i]!='\0')
{
        if((s[i] >= 'A' && s[i]<='Z') || (s[i]>='0' && s[i]<='9'))
        {
                if(s[i] >= 'A' && s[i]<='Z') cn = s[i]-65;
                else cn = (s[i]-48)+26;
                switch(cs)
                {
                        case CRE:
                        SDL_BlitSurface(arr_cre[cn],NULL,surface,&dstrect);
                        dstrect.x += 9;
                        break;
                        case CYE:
                        SDL_BlitSurface(arr_cye[cn],NULL,surface,&dstrect);
                        dstrect.x += 9;
                        break;
                        case CRB:
                        SDL_BlitSurface(arr_crb[cn],NULL,surface,&dstrect);
                        dstrect.x += 17;
                        break;

                }
        }
        i++;
}
```

3.2.5.2. Menú de inicio

El motivo de implementar un menú de inicio es para darle un envoltorio al juego. La opción más importante del menú es la de jugar una partida (PLAY) y salir (EXIT), ya que lo demás ya viene explicado en el fichero README.

Cuando el jugador selecciona exit no se le pregunta si de verdad desea salir ya que esto tendría más sentido en un juego más pesado que tardase más en cargar. Si el jugador sale por error de nuestro juego podrá cargarlo rápidamente de nuevo.

En el menú podemos resaltar las opciones con los cursores y aceptar una opción con la tecla RETURN.

3.2.5.3. Configuración de las teclas

En la etapa de diseño había pensado en un fichero de configuración o en implementar la opción de cambiar los controles desde el juego. Finálmente la personalización de las teclas se puede hacer editando el fichero cop.h a modo de fichero de configuración.

3.2.5.4. Mejores puntuaciones

He suprimido esta carácterística durante la implementación, ya que ahora mismo aporta poco al juego y ya tenemos el menú de inicio, los marcadores y la pantalla de puntuación final para proporcionar una presentación aceptable.

3.2.5.5. Puntuación final

Al acabar la partida se le muestra al jugador la suma total de los puntos obtenidos. Cada pandillero vale 10 puntos, a los cuales se les suma a modo de bonificación el tiempo transcurrido desde el inicio de la partida. Una vez que el jugador ha sido informado de su puntuación, puede volver al menú de inicio presionando ESC o RETURN.

3.2.6. Sonido

Con SDL_mixer se han cubierto los requerimientos básicos de sonido.

Durante la inicialización del programa cargamos nuestros ficheros .wav en memoria y SDL_mixer se encarga de gestionarlos a través de varios canales de reproducción simultanea. Lo cual es justo lo que necesitamos para escuchar sirenas de policia, gritos y disparos a la vez.

```
32 Mix_Chunk *chunks[NCHUNKS];
33
34 int busy_ch[NCHUNKS];
35
36 void loadChunks(void)
37 {
38         int chunk = 0;
39
40         while(chunk < NCHUNKS)
41         {
42                 chunks[chunk] = Mix_LoadWAV(samplesd[chunk]);
43                 chunk++;
44         }
```

4. Código fuente

La licencia para el código fuente es la GNU Lesser General Public License versión 2.1, también adoptada por SDL: http://www.libsdl.org/license-lgpl.php.

En el código fuente se incluye una copia.

5. Conclusión

El nombre elegido finálmente para el videojuego es “Muggers”, que significa atracadores o delincuentes en inglés.

La versión 0.1 cumple con los principales objetivos propuestos inicialmente en la etapa de diseño, y a nivel de gráficos y jugabilidad tenemos un desarrollo rápido de la acción , con gran cantidad de enemigos en pantalla, sonido envolvente de gritos y sirenas de policia, y respuestas visuales y auditivas a los disparos para enganchar a los jugadores.

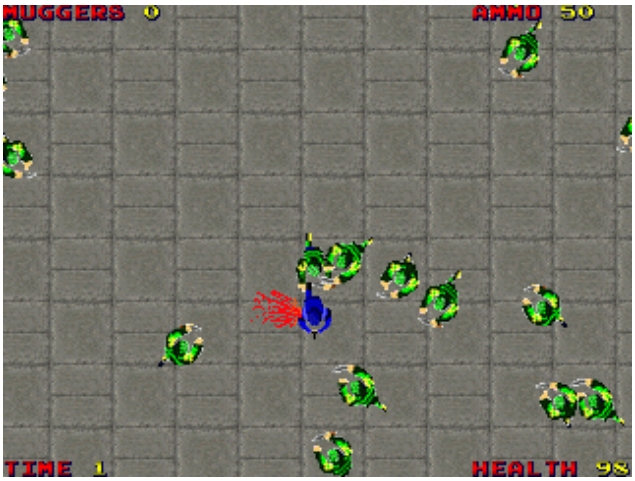
A partir de aquí se podría continuar con el desarrollo para añadir algunas de las mejoras propuestas en el primer apartado de la memoria empezando por la organización de los gráficos y sus funciones de carga (fue de lo primero que hice) , u hospedarlo ya en alguna web para mostrarlo a los jugadores o en busca de colaboradores si decido mantenerlo.

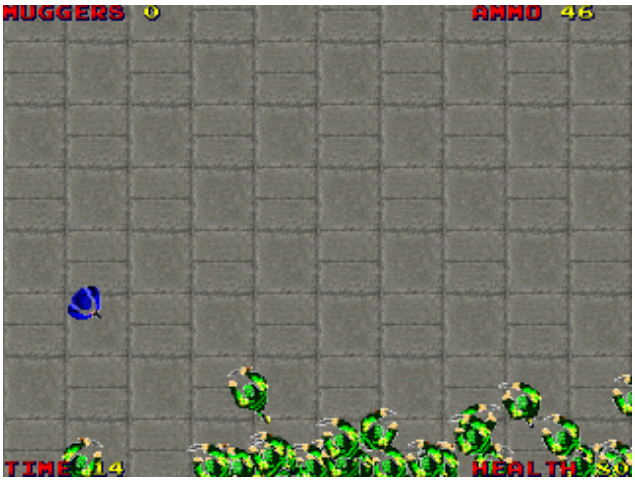
6. Referencias

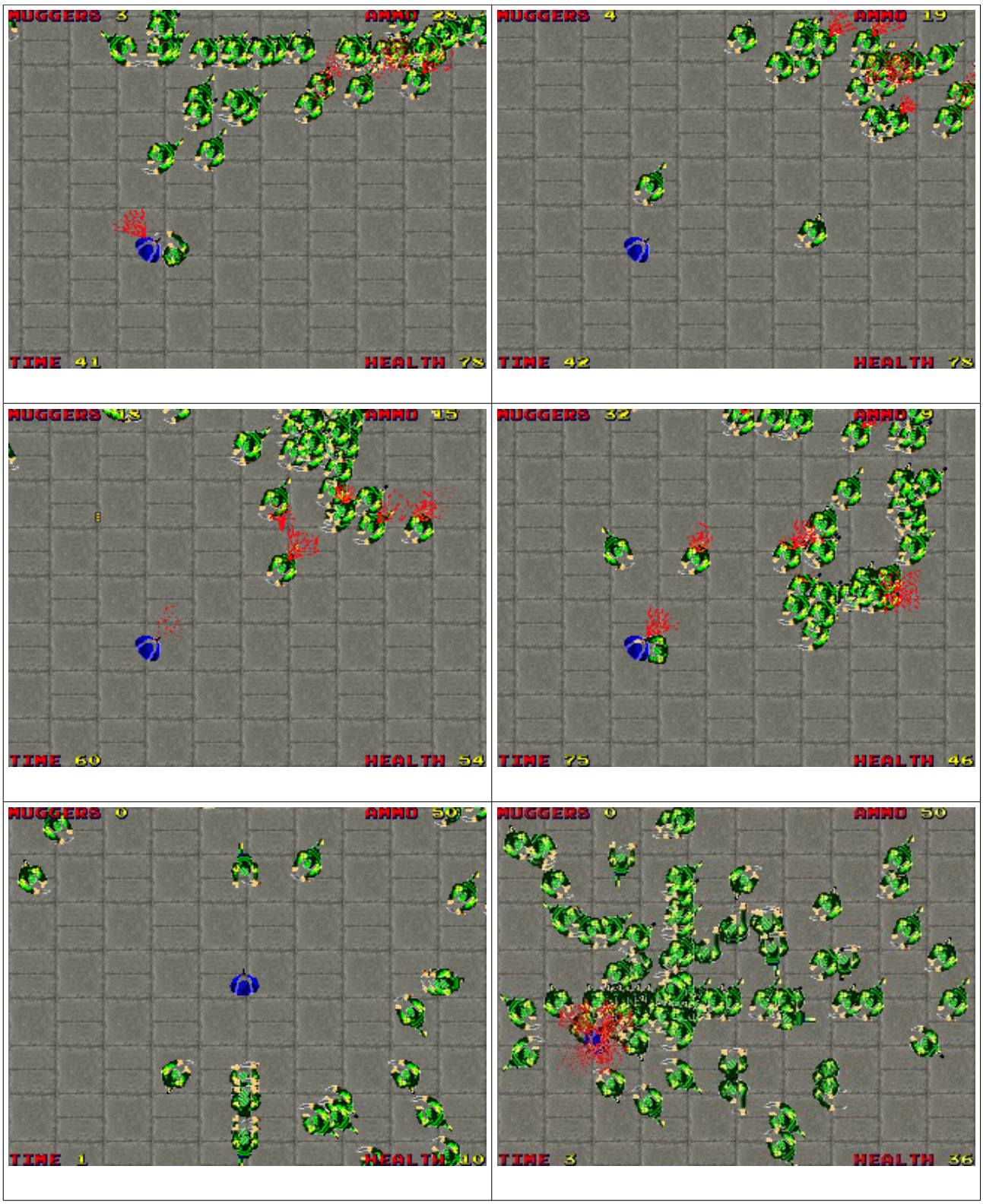
- Brian Kernighan, Dennis Ritchie. El lenguaje de programación C, 2ª edición, Prentice Hall, 1988.
- http://gcc.gnu.org/
- http://www.gnu.org/software/make/
- http://www.libsdl.org
- http://www.libsdl.org/projects/SDL_mixer/
- http://projects.gnome.org/gedit/
- http://www.ubuntu.com
- http://www.gimp.org/
- http://www.wikipedia.org

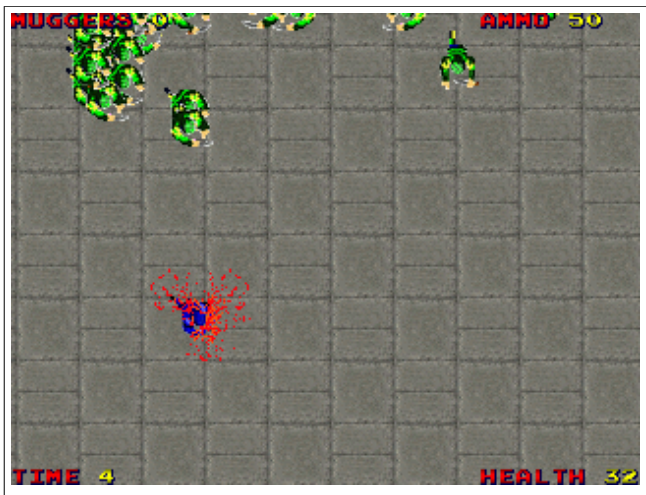
Apendice A

Algunas capturas de pantalla de “Muggers”:

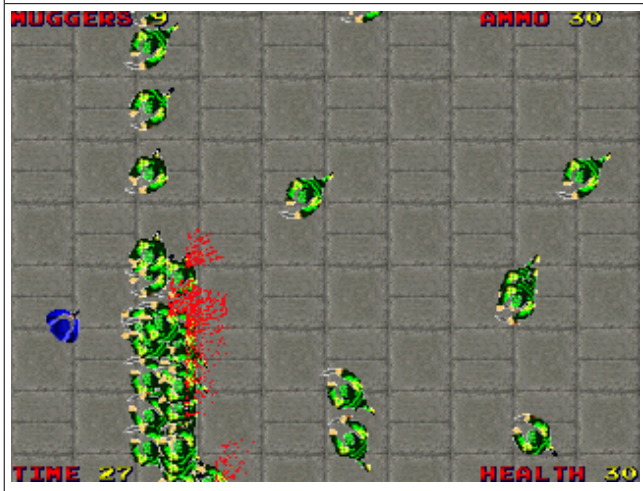




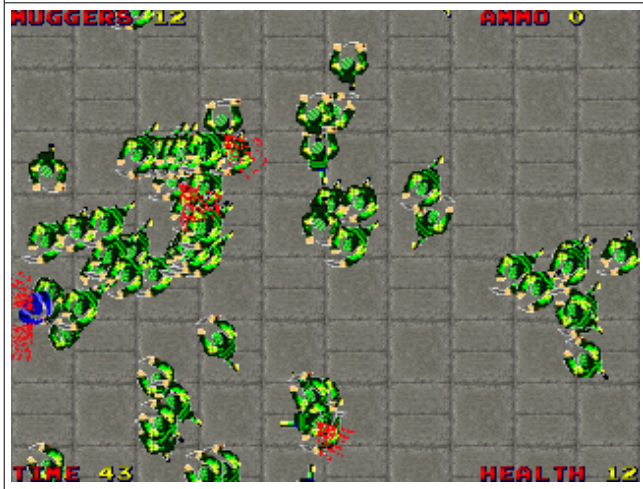

MUGGERS 3
AMMO 28
TIME 41
HEALTH 78
MUGGERS 4
AMMO 19
TIME 42
HEALTH 78
AMMO 15
TIME 60
HEALTH 54
MUGGERS 32
TIME 75
HEALTH 46
MUGGERS 0
AMMO 50
TIME 1
MUGGERS 0
AMMO 50
TIME 3
HEALTH 36


AMMO 50
TIME 4
HEALTH 32

MUGGERS 9
AMMO 40
TIME 26


MUGGERS 9
AMMO 30
TIME 27
HEALTH 30

MUGGERS 10
AMMO 39
TIME 38
HEALTH 18


MUGGERS 12
AMMO 0
TIME 43
HEALTH 12

MUGGERS 16
AMMO 21
TIME 47
HEALTH 4