

Lagunex

**Agustín Rodríguez González de Aguilar
Ramón Rodrigo Luque Luque**

Índice

	Página
Introducción.....	2
Descripción.....	2
Métodos y herramientas.....	5
Análisis de aplicación.....	6
Diseño.....	6
Conclusiones.....	9
Mejoras.....	9
Referencias.....	10

Introducción

Lagunex es un proyecto con el que intentamos hacer un juego de plataformas en 2D que pudiese ser controlado mediante teclado y joystick, para hacer la emulación de los juegos que hemos conocido que se jugaban en máquinas.

La finalidad de este proyecto es la introducción por nuestra parte en la programación de videojuegos en su fase más básica (un sólo jugador y programación en 2D) y así comprender la problemática que puede existir al hacer futuros proyectos que abarquen otros puntos como podrían ser juegos en 3D o que se enlacen mediante la red y puedan jugar varios a la vez.

La temática que hemos utilizado es la del propio Instituto donde estamos estudiando en la que hemos incluido un personaje principal que deberá ir pasando por diferentes pantallas y niveles y al cuál le ponen trabas distintos enemigos y estructuras creadas.

Descripción

Se ha tratado de incluir una serie de enemigos y estructuras que irán dificultando el paso del personaje por las distintas pantallas y niveles. El objetivo del personaje será llegar al final de cada uno de los niveles en los que se enfrentará a un enemigo más difícil de eliminar y tras es cuál pasará al siguiente nivel. Existen cuatro niveles del juego con cuatro enemigos difíciles (“boss”) que son nuestros profesores una vez superados todos estos se finalizará la función básica del juego.

Incluimos en la documentación del juego una herramienta libre que permitirá al usuario, siguiendo una pequeña guía, la creación de sus propios niveles de juego y cuya dificultad planteará él mismo.

Pasamos a describir los diferentes enemigos y estructuras que se podrán encontrar:

1. Zonas de riesgo: Al caer en una zona de riesgo el personaje comenzará desde el último punto de paso.



2. Plataformas móviles: Estas servirán de apoyo al personaje para su paso a lugares inaccesibles mediante el salto o desplazamiento normal.



3. Zonas deslizantes: Son zonas en las que se realiza una efecto de deslizamiento cuando el jugador intenta frenar.



4. Enemigo de desplazamiento lateral: Es un personaje que se desliza entre un rango de espacio y cuya colisión del personaje saltando sobre él provoca su eliminación.



5. Enemigo de persecución: Se desplaza hacia el lugar donde se encuentre el personaje principal, pero no tiene la propiedad de saltar, por lo que cuando encuentra un escalón se queda atrapado. Tampoco detecta que puede caer al vacío. Al igual que el de desplazamiento lateral puede ser eliminado saltando sobre él.



6. Enemigo lanzador: Es una estructura fija que realiza el lanzamiento de balas cuya colisión con el personaje lo eliminan. La colisión del personaje principal con estas estructuras no produce ningún evento, sin embargo sí que pueden ser destruidas con las balas del personaje. Si el personaje salta sobre cualquiera de las balas que lanza o las colisiona con la cabeza puede destruir dichas balas sumando así una serie de puntos.



7. Enemigo fin de nivel (“boss”): Es un enemigo que consta con mayor número de vidas que los demás, por lo tanto su dificultad de eliminación es superior. Pero, por contra, tenemos que cuando éste aparece comienzan a caer de forma aleatoria municiones que permitirán reponerse y así destruirlo. Existen cuatro diferentes los cuales a modo de homenaje hemos puesto la cara de nuestros profesores, a mayor nivel mayor dificultad de eliminación.



8. Vidas y munición: Son estructuras que nos ayudarán a superar el nivel y a proveernos de balas necesarias.



9. Personaje principal: Es el personaje protagonista que manejaremos.



10. Explosión: Es la imagen que aparecerá cuando es destruido un NPC o una bala.



Métodos y Herramientas

El hardware usado para la programación de este juego es un equipo Inter Pentium con doble núcleo a 3.00 Ghz.

Hemos usado como entorno de programación el IDE Codeblocks de distribución libre y como sistema operativo gnu/linux (distro ubuntu) y Windows XP.

El lenguaje usado para la implementación de las funciones es el C/C++ con librerías de SDL.

Hemos utilizado en Windows el compilador “MinGW”, herramienta libre, y en sistema linux el “gcc”.

Para la creación de los mapas que se usan en el juego hemos usado una herramienta libre llamada “Mappy”.

Para la modificación de las imágenes hemos utilizado el programa “GIMP 2.0” de distribución libre.

Para la búsqueda a través de Internet hemos usado preferentemente el navegador “Mozilla Firefox”

Análisis de aplicación

Inicialmente el juego fue pensado para ampliar conocimientos sobre los lenguajes de programación y nos planteamos que este sería un buen reto debido a que existen varios tipos de novedades:

- Nunca habíamos trabajado con imágenes gráficas en programación.
- La repetición casi continua del dibujado nos haría pensar en métodos de optimización del rendimiento.
- Era un proyecto totalmente novedoso para nosotros.

La especificación de requisitos para el correcto funcionamiento del juego serán en estos apartado:

- Software:
 - Sistema operativo Gnu/linux o sistema Windows XP o superior.
- Hardware:
 - Procesador con frecuencia superior a 800mhz.
 - Memoria ram: 128mb. o superior
 - Raton/teclado (Compatibilidad con joyStick).
 - Tarjeta de sonido.

Diseño

Hemos utilizado una serie de estructuras que a continuación detallamos:

ESTRUCTURAS DE RECUPERADO

Son estructuras creadas para hacer la recuperación del fondo de pantalla una vez que se ha dibujado los que necesitásimos de “tiles”, personajes, NPCs, balas y explosiones. Un “tile” define las unidades básicas de dibujado en pantalla que hemos asignado a un valor de 32 pixels de ancho y 24 de alto. Corresponden a lo que sería

un ladrillo en la pantalla. La asignación de estos valores es porque queríamos en pantalla un mapa de 24x24 ladrillos (tiles) y éstos valores eran divisores de la pantalla a utilizar (800x600).

struct tile: En esta estructura guardamos los “tiles” dibujados en pantalla para luego ser recuperados con el fondo para su redibujado.

struct tileNPC: En esta estructura guardamos las posiciones donde son dibujados los NPCs (Non Playable Characters) para luego recuperarlos como fondo en el redibujado.

struct tilepersonaje: Posición del personaje para en pantalla para su recuperación.

struct tilebalaspropias: Aquí guardamos las posiciones donde han sido dibujadas las balas propias para ser recuperadas después del dibujado.

struct tilebalasenemigas: Aquí guardamos las posiciones donde han sido dibujadas la balas de los distintos NPCs para ser recuperadas.

struct tileexplosiones: Estructura que almacena las posiciones de las explosiones que se producen para recuperar el fondo de pantalla tras su dibujado.

DIBUJADO DE IMÁGENES

struct tipo_cuadro: En esta estructura definimos la posición de un rectángulo dando su posición x, y, el ancho y el alto.



struct tipo_cuadro_dibujo: Definimos tres cajas que conformarán los límites de colisión de cualquiera de los personajes (NPCs, tiles, balas, ...) que aparecerán, des esta forma hacemos un poco más real nuestro movimiento y roce con los demás componentes del juego, aunque complica bastante el sistema de colisionado.



class Sprite: Esta clase se encarga del manejo de las imágenes y

tiene funciones de carga, división de cada imagen en cada uno de los frames (imagen por separado) que la componen, asignación de los cuadros para su colisión y asignación del tipo de imagen al tipo de componente que sea (personaje principal, NPC, bala, ...).

class Tipo_personaje: Esta clase se encarga de todo lo relativo al personaje principal: movimiento, colisión con balas, NPCs, tiles, ... y de llevar el computo de vidas y puntos.

class Tipo_NPC: Esta clase se encarga de todo lo relativo a los NPCs: movimiento, colisión entre ellos, con los tiles y las balas y del control de periodo de disparo de los NPCs que tengan esta propiedad.

class Tipo_Balas: Esta clase se encarga del movimiento de las balas y colisión con los tiles.

struct Tipo_Explosion: Esta estructura sólo se encarga de saber si una explosión en particular está activa y de cual será su posición de dibujado.

class Barra_Control: Maneja el control de la barra de sonido de música y efectos.

struct bala_en_espera: Determinan los datos a guardar de las balas que creamos de los NPCs y que se mantienen en espera hasta hay hueco y que pueden ser creadas.

struct tipo_colision: Estructura que define si hay o no colisión por los distintos lados de un personaje (superior, inferior, derecha o izquierda)

class clase_colision: Esta clase implementa las funciones necesarias para determinar si hubo o no colisión entre dos estructuras de cajas compuestas de tres cajas pasadas como parámetros.

MÚSICA

class Mus_Efec: Implementa las funciones necesarias para la reproducción y detención de la música y los efectos sonoros.

TIPOS ABSTRACTOS DE DATOS

struct tipoCola: Guarda la información de un nodo de una estructura de tipo cola (FIFO) mediante asignación de memoria dinámica.

class clase_cola: Implementa las funciones necesarias para el manejo de una cola (FIFO) utilizando la estructura tipo_cola.

struct elemento: Define el nodo de una lista simplemente enlazada con contenido de datos genérico (void *).

class Clase_Lista: Implementa las funciones necesarias para el manejo de la lista simplemente enlazada.

OTROS

struct Raton: Define la posición del ratón en la pantalla y si se ha pulsado el botón izquierdo.

Conclusiones

En principio no se ha conseguido todos los objetivos que se pretendían en una primera exposición de lo que se deseaba ya que el dotar a los NPCs de cierta autonomía implicaba una estrategia de inteligencia artificial muy básica que nos habría llevado demasiado tiempo, aunque se han mejorado ciertos aspectos de realidad que inicialmente no se habían tenido en cuenta como es el caso de las colisiones y la suavización de movimientos del personaje que dan apariencia de movimiento casi real.

Mejoras

Se han detectado un par de fallos en el colisionado que se deben arreglar.

También para próximas versiones se podrían incluir varias mejoras como:

- dos jugadores
- jugar a través de red.
- cierta inteligencia artificial en los NPCs.

Referencias

Referencia bibliográfica:

Programación de Videojuegos en SDL. Autor: Alberto García Serrano

Referencia web:

<http://www.libsdl.org> : Librerías gráficas SDL

<http://www.codeblocks.org> : Entorno de programación

<http://www.mingw.org> : Compilador C/C++ para windows

<http://gcc.gnu.org> : Compilador C/C++ para linux

<http://www.mappy.com/espanol> : Creador de niveles

<http://www.gimp.org.es> : Tratamiento gráfico

<http://www.deviantart.com> : Algunas imágenes

<http://supertux.lethargik.org> : Algunas texturas del juego con licencia GLP "Super Tux"

<http://www.mozilla-europe.org/es/firefox> : Navegador de búsquedas en la red

También hemos tomado información y datos de imagen y sonido de otras páginas buscadas en la red, aunque no hemos llegado a anotarlas.