

Código Fuente

Cliente

```
#ifndef ANGULO_H_
#define ANGULO_H_
#include "constantes.h"

class Angulo
{
public:
    static void corregir(long* valor);
    /* Pasajes */
    static double aRadianes(long grados);
    static long aGrados(double radianes);
};

#endif /*ANGULO_H_*/
```

```
#ifndef BARRA_H_
#define BARRA_H_

#include "ElementoLongitudinal.h"

class Barra : public ElementoLongitudinal {

public:
    Barra();
    Barra(Coordenada posicion, ulong magnitud, long angulo, long ancho);
    virtual ~Barra();
};

#endif /*BARRA_H_*/
```

```
#ifndef BOTON_H_
#define BOTON_H_

#include "PanelContenedor.h"
#include "Mapa.h"
#include <string>

class Boton : public PanelContenedor
{
private:
    /* Rutinas */
    void (*pf_rutinaMouseDown)(Coordenada, Boton * );
    void (*pf_rutinaMouseUp)(Coordenada, Boton * );
    void (*pf_rutinaMouseOver)(Coordenada, Boton * );

    /* Buscadores de imagenes */
    std::string getRutaImgDefecto(int nroDeBoton);
    std::string getRutaImgMouseDown(int nroDeBoton);
    std::string getRutaImgMouseOver(int nroDeBoton);

    /* Referencia al mapa sobre el cual interactua */
    Mapa * mapa;
    /* Indica si esta presionado */
    bool presionado;

public:
    /* Constructores y destructores */
    Boton(int nroDeBoton, Mapa *);
    virtual ~Boton();
    /* Eventos del mouse */
    void evento_MouseDown(Coordenada punto);
    void evento_MouseUp(Coordenada punto);
    void evento_MouseOver(Coordenada punto);
    /* Setters de eventos */
    void setRutinaMouseDown(void (*rutina)(Coordenada, Boton *));
    void setRutinaMouseUp(void (*rutina)(Coordenada, Boton *));
    void setRutinaMouseOver(void (*rutina)(Coordenada, Boton *));
    /* Setters, getters y Envios de seniales */
    bool esta_presionado() const;
    void desapretar();
    void enviarSenialSalida();
    void enviarSenialSms();
    void enviarSenialSimulacion();
    long pedirTiempoResolucion();
    bool getSmsState();
    Mapa * getMapa() const;
};

#endif /*BOTON_H_*/
```

```
#ifndef CINTA_H_
#define CINTA_H_

#include "ElementoLongitudinal.h"

class Cinta : public ElementoLongitudinal {
public:
    Cinta();
    Cinta(Coordenada posicion, ulong magnitud, long angulo, long ancho);
    virtual ~Cinta();
};

#endif /*CINTA_H_*/
```

```
#ifndef __CHAT_DOBLE_H__
#define __CHAT_DOBLE_H__


#include <string>
#include <vector>
#include <map>
#include <SDL.h>
#include "SDL_image.h"
#include <SDL_opengl.h>
#include "common_Mutex.h"

#define MAX_LINEAS 13
#define MAX_CARACTERES 20

class ChatDoble
{

private:
    std::vector<std::string> lineas; //Guarda las lineas del chat
    std::string buffer; //Guarda lo que está escribiendo el usuario
    std::vector<std::string> entradas; //Guarda las entradas ya ingresadas por el usuario
    Mutex m;
    std::map<char,GLuint> texturas; //texturas disponibles

    bool visible; //se ve el chat
    bool salir;

    bool simular; //debe simular
    bool bloquear; //se debe bloquear al usuario

    bool nuevoMovimiento; //hay nuevo movimiento disponible
    std::string movimiento; //mapa rival

    /* Carga textura */
    void loadTexture(const char * path, char letra);

    /* Imprime el fondo del chat */
    void imprimirFondo();

    /* Imprime un caracter en la posición indicada */
    void imprimirCaracter(char letra, float x, float y);


public:
    /* Constructor */
    ChatDoble();

    /* Agrega una letra.
     * Devuelve false si el buffer está lleno */
    bool agregarLetra(char letra);

    // backspace
    bool quitarLetra();

    // enter del chat
    bool enter();

    /* Agrega linea al chat, pisa una si hace falta */
    void agregarLinea(const std::string& linea);

    /* Imprime todo el chat */
    void imprimirChat();

    /* Obtener entrada del usuario. False si no hay más */
    bool obtenerEntrada(std::string& entrada);

    /* Devuelve una letra si la mapea. -1 si no lo hace */
    char esLetra(SDL_Event evento);

    /* Carga las texturas del chat */
    void cargarTexturas();
```

```
/* Modifica el valor de la visibilidad del chat */
void cambiarVisible();

/* Pregunta si la ventana debe salir */
bool debeSalir();

/* Modifica el valor de salir */
void setSalir(bool valor);

/* Carga el mapa del rival en xml.
 * Devuelve true si hay mapa disponible, false si no */
bool obtenerMovimiento(std::string& xml);

/* Modifica el movimiento que se está almacenando */
void cambiarMovimiento(const std::string& xml);

/* Imprime el fondo del mapa rival */
void imprimirFondoRival();

/* Indica si el Servidor mandó a simular */
bool debeSimular();

/* Modifica el valor de simular */
void setSimular(bool valor);

/* Indica si debe bloquear los botones */
bool debeBloquear();

/* Modifica el valor de bloquiar */
void setBloquear(bool valor);

/* Agrega un mensaje a las entradas */
void agregarMensaje(const std::string& mensaje);

};

#endif
```

```
#ifndef __CLIENT_CONTACTA_SERVER_H__
#define __CLIENT_CONTACTA_SERVER_H__


#include "common_SocketCliente.h"
#include <vector>
#include "common_Thread.h"
#include <iostream>

class ContactaServer: public Thread
{

private:
    std::vector<std::string> mensajesRecibidos; //Mensajes recibidos desde el server
    bool conectado; //Indica si el socket está conectado o no
    SocketCliente socket; //Socket de contacto con el server

    /* Modifica el valor de conectado */
    void setConectado(bool valor);

public:

    /* Constructor. Crea el Socket por el que se contactará al Server */
    ContactaServer();

    /* Destructor */
    virtual ~ContactaServer();

    /* Analiza si el socket está conectado o no */
    bool socketConectado();

    /* Función que ejecuta el hilo.
     * Espera mensajes del Servidor y los va ubicando en una lista
     * para que sean posteriormente analizados */
    void run();

    /* Intenta conectarse con el host y el puerto dato.
     * Retorna true si se pudo conectar */
    bool conectar(const std::string& host, const int puerto);

    /* Devuelve el primer mensaje que se recibió que esté disponible.
     * Devuelve false si no hay ninguno */
    bool obtenerMensajeRecibido(std::string& mensaje);

    /* Envía un mensaje a través del socket */
    bool enviarMensaje(const std::string& mensaje);

    /* Desconecta al socket */
    void desconectarSocket();

    /* Agrega un mensaje recibido a la lista */
    void agregarMensaje(const std::string& mensaje);

    /* Vacía el vector de mensajes recibidos */
    void vaciarMensajesRecibidos();

};

#endif
```

```
#ifndef __CHAT_HILO_MOVIMIENTOS_H__
#define __CHAT_HILO_MOVIMIENTOS_H__

#include <string>
#include "common_Thread.h"
#include "Mapa.h"

class HiloMovimientos: public Thread
{
    private:
        std::string xmlListo;
        bool listo;

        Mapa* mapa; //referencia al mapa

    public:
        /* Constructor. Recibe referencia al mapa */
        HiloMovimientos(Mapa* mapa);

        /* Se va simulando el retardo */
        void run();

        /* Devuelve true si hay un movimiento disponible y lo carga en el parámetro */
        bool obtenerMovimiento(std::string& xml);
};

#endif
```

```
#ifndef __INTERFAZ_BATALLA_H__
#define __INTERFAZ_BATALLA_H__


#include "client_ContactaServer.h"
#include "client_HiloMovimientos.h"
#include "Mapa.h"
#include "common_Thread.h"
#include "client_ChatDoble.h"
#include <vector>
#include <string>
#include <glib.h>
#include <glib/gprintf.h>
#include <gtk/gtk.h>

class InterfazBatalla: public Thread
{
    private:

        ContactaServer* contacto; //contacto con el servidor
        Mapa* mapa; //referencia al mapa
        ChatDoble* chat; //chat que se usa de puente con la interfaz gráfica
        HiloMovimientos* movimientos; //hilo que simula el retardo de los movimientos

        int puntos;

        bool jugando;

        //desconvierte el hexadecimal
        static char obtenerValor (char hexa);

    public:
        //decodifica el mensaje en hexadecimal
        static void desconvertir(std::string& resultado, const std::string& original);

        /* Constructor */
        InterfazBatalla(ContactaServer* contacto);

        /* Prepara el proceso para la batalla. Envía los archivos */
        void iniciarBatalla(ChatDoble* chat);

        /* Analiza el tipo de mensaje recibido y actúa en consecuencia */
        bool analizarMensajesRecibidos();

        /* Devuelve true si la batalla está en curso */
        bool estaJugando();

        /* Devuelve los puntos que realizó */
        int getPuntos();

        /* Modifica el valor de jugando */
        void setJugando(bool valor);

        /* Se efectúa la batalla */
        void run();

};

#endif
```

```
#ifndef __INTERFAZ_GRAFICA_H__
#define __INTERFAZ_GRAFICA_H__


#include <vector>
#include <glib.h>
#include <glib/gprintf.h>
#include <gtk/gtk.h>
#include "common_Thread.h"
#include "client_ContactaServer.h"
#include "client_InterfazBatalla.h"
#include "client_InterfazJugarSolo.h"
#include "client_ChatDoble.h"
#include <iostream>
#include <string>

class InterfazGrafica: public Thread
{
    private:

        /* Elementos de la ventana */
        GtkWidget* ventana;
        GtkWidget* window;
        GtkWidget* entrada;
        GtkWidget* etiqueta;
        GtkWidget* cuadroMensajes;

        /* Contacto con el servidor */
        ContactaServer contacto;

        /* Interfaz para juego individual */
        InterfazJugarSolo* juegoSolo;

        /* Usuarios rivales */
        std::string adversarios;
        bool adversariosActualizados;

        std::vector<std::string> mensajesLeidos; //Mensajes que fueron escritos por el usuario

        /* Elementos de las batallas */
        bool enBatalla;
        InterfazBatalla* batalla;
        ChatDoble* chat;

        std::vector<std::string> mensajesAMostrar; //Se deben mostrar en el cuadro de mensajes

        /* Se invoca al emitirse la señal clicked del botón */
        static void apretar_boton_aceptar(GtkWidget *widget, gpointer data);

        /* Se invoca al emitirse la señal delete_event */
        static gboolean on_delete_event(GtkWidget *widget, GdkEvent *event, gpointer data);

        /* Se invoca al emitirse la señal destroy */
        static void destruir(GtkWidget *widget, gpointer data);

        /* Se invoca cada 100 milisegundos */
        static gboolean timeout_handler(gpointer data);

        /* Analiza si el texto está vacío o no
           Se considera vacío si es de longitud cero o solo tiene espacios */
        static bool textoVacio(const std::string& texto);

        /* Analiza cada uno de los mensajes recibidos y actúa en consecuencia de su tipo */
        static void analizarMensajesRecibidos(InterfazGrafica* p);

        /* Se ejecuta al hacer click en el botón conectar. Abre la ventana para setear datos
           de conexión */
        static void apretar_boton_conectar(GtkWidget *widget, gpointer data);

        /* Intenta conectar con el servidor con los datos de la ventana de conexión */
        static void conectar(GtkWidget *widget, gpointer data);
```

```
/* Cierra la ventana dada por parámetro */
static void cerrar_ventana(GtkWidget *widget, gpointer data);

/* Se desconecta del Servidor */
static void desconectar(GtkWidget *widget, gpointer data);

/* Se ejecuta al apretar en el botón Jugar Con.
 * Pide los usuarios disponibles y permite elegir al usuario a quien desafiar */
static void jugar_batalla(GtkWidget *widget, gpointer data);

/* Obtiene los tokens de la cadena dada */
static void obtenerTokens(std::string cadena, std::vector<std::string>& tokens);

/* Muestra un GtkDialog con la leyenda dada por encima de la ventanaPadre */
static void mostrarAlerta(std::string leyenda, GtkWidget* ventanaPadre);

/* Envía solicitud de juego doble con el usuario elegido en la lista */
static void elegirAdversario(GtkWidget *clist, gint row, gint column, GdkEventButton
*event, gpointer data);

/* Se recibió una propuesta de batalla.
 * Consulta al usuario si desea jugar con esta persona o no */
static void propuestaBatalla(InterfazGrafica* pthis, const std::string& rival);

/* Envía notificación de rechazo de batalla */
static void rechazarBatalla(GtkWidget *widget, gpointer data);

/* Inicia una partida doble. Setea todo lo necesario para el comienzo de una interfaz
batalla */

static void iniciarBatalla(InterfazGrafica* pthis);

/* Envía notificación de aceptación de batalla */
static void aceptarBatalla(GtkWidget *widget, gpointer data);

/* Inicializa una partida individual */
static void jugar_solo(GtkWidget *widget, gpointer data);

/* Función que se invoca al pulsar un botón utilizado para pruebas durante el
desarrollo */
static void apretar_boton_conectar_auto(GtkWidget *widget, gpointer data);

/* Se invoca después de un tiempo una vez al principio del programa una sola vez.
 * Cierra la presentación */
static gboolean inicio(gpointer data);

/* Genera la ventana que permite modificar la gravedad */
static void modificar_gravedad(GtkWidget *widget, gpointer data);

/* Modifica el archivo de configuración */
static void modificar_archivo_gravedad(GtkWidget *widget, gpointer data);

/* Pasa de una entero a un string */
static void enteroAString(std::string& cadena, ulong entero);

/* Muestra el resultado de un juego individual y consulta si desea seguir jugando o no
*/
static void resultado_juego_solo(InterfazGrafica* pthis, const std::string& puntos);

/* Pide al Servidor una tabla de posiciones */
static void pedir_posiciones(GtkWidget *widget, gpointer data);

public:

/* Constructor */
InterfazGrafica();

/* Hilo que crea la interfaz gráfica y la ejecuta */
void run();

/* Carga un mensaje leído en el parámetro.
 * Devuelve false si no hay mensajes leídos */
```

```
bool obtenerMensajeLeido(std::string& mensaje);  
/* Agrega un mensaje para mostrar en el cuadro de texto de la interfaz */  
void agregarMensajeAMostrar(const std::string& mensaje);  
};  
#endif
```

```
#ifndef __INTERFAZ_JUGAR_SOLO_H__
#define __INTERFAZ_JUGAR_SOLO_H__

#include "common_Thread.h"
#include "Mapa.h"

class InterfazJugarSolo : public Thread
{
    private:
        bool jugando; //True si está jugando
        int puntos;
        Mapa* mapa; //Referencia al mapa

        /* Modifica el valor de jugando */
        void setJugando(bool valor);

    public:
        /* Constructor. Recibe una referencia al Mapa */
        InterfazJugarSolo(Mapa* mapa);

        /* Indica si está jugando o no */
        bool estaJugando();

        /* Encapsula la edición del escenario */
        void run();

        /* Devuelve los puntos que hizo el usuario en la partida */
        int getPuntos();
};

#endif
```

```
#ifndef COHETE_H_
#define COHETE_H_

#include "ElementoLongitudinal.h"

class Cohete : public ElementoLongitudinal
{
public:
    Cohete();
    Cohete(Cordenada posicion, ulong magnitud, long angulo, long ancho);
    virtual ~Cohete();
};

#endif /*COHETE_H_*/
```

```
#ifndef __MUTEX_H__
#define __MUTEX_H__

#include <pthread.h>

class Mutex
{
    private:
        pthread_mutex_t mut;

        /* Constructor copia privado */
        Mutex(const Mutex& m);

        /* Operador = privado */
        Mutex& operator = (const Mutex& original);

    public:
        /* Constructor */
        Mutex();

        /* Lockea el mutex */
        void lock();

        /* Deslockea el mutex */
        void unlock();
};

#endif
```

```
#ifndef __COMMON_SOCKET_H__
#define __COMMON_SOCKET_H__


#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <iostream>

#define SOCKET_INVALIDO -1
#define BACKLOG 10
#define TAM_BUFFER 256

class Socket
{

private:
    int id; //Identificador del Socket
    sockaddr_in dir; //Mi dirección
    char historial[TAM_BUFFER + 1]; //Bytes remanentes del recv
    unsigned int validosHistorial; //Cantidad de bytes remanentes del recv

public:
    /* Constructor */
    Socket();

    /* Destructor */
    virtual ~Socket();

    /* Crea al socket. Devuelve true si fue una creación exitosa. */
    bool create();

    /* Envía a través del Socket el dato indicado.
     * Devuelve true si pudo enviar correctamente */
    bool send(const std::string& dato) const;

    /* Recibe un mensaje a través del Socket y lo almacena en dato.
     * Devuelve true si pudo recibir correctamente */
    bool recv(std::string& dato);

    /* Modifica el Id del Socket */
    void setId(const int id);

    /* Devuelve el Id del Socket */
    int getId() const;

    /* Devuelve mi dirección */
    sockaddr_in& getDir();

    /* Cierra el Socket */
    void close();

    void shutdown(int modo);

};

#endif
```

```
#ifndef __SOCKET_CLIENTE_H__
#define __SOCKET_CLIENTE_H__


#include "common_Socket.h"
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <iostream>

class SocketCliente: public Socket
{
public:
    /* Trata de conectarse a un Socket que se encuentra escuchando
     * en la dirección y puerto indicados.
     * Devuelve true si fue una conexión exitosa. */
    bool connect(const std::string direccion, const int puerto);
};

#endif
```

```
#ifndef __THREAD_H__
#define __THREAD_H__

#include <pthread.h>
#include "common_Mutex.h"

class Thread
{
    private:
        pthread_t hilo;
        Mutex mHilo; //Evita que se lean y escriban los atributos al mismo tiempo.
        bool vivo; //True si tiene que seguir corriendo
        bool corriendo; //True si el hilo está corriendo

        /* Constructor copia privado */
        Thread(const Thread& t);

        /* Operador = privado */
        Thread& operator = (const Thread& original);

    public:
        /* Constructor */
        Thread();

        /* Destructor */
        virtual ~Thread();

        /* Le indica al hilo que debe suicidarse */
        void terminar();

        /* Devuelve true si el hilo aún está corriendo */
        bool estaCorriendo();

        /* Método estático que se empieza a correr al crearse el hilo de tipo pthread */
        static void* static_run (void* p);

        virtual void run() = 0;

        /* Espera que el hilo termine su ejecución */
        void join() const;

        /* Devuelve true si el hilo debe seguir vivo */
        bool estaVivo();

        /* Método que inicia la ejecución del hilo */
        void start();

    protected:
        /* Bloquear y desbloquear estado se utilizan para evitar que se acceda
         * a un atributo del hilo al mismo tiempo (desde el hilo mismo y desde el
         * hilo que lo creó */
        void bloquearEstado();
        void desbloquearEstado();

        /* Modifica el valor de corriendo */
        void setCorriendo(bool valor);

};

#endif
```

```
#ifndef CONFIG_FILE_H
#define CONFIG_FILE_H

#include "config.h"
#include <map>

class ConfigFile : public Config
{
    typedef std::map<std::string, std::string> TStrStrMap;
    TStrStrMap data_;

protected:
    virtual std::string readString(const std::string& key) const;

public:
    ConfigFile(const std::string& filename);
    virtual ~ConfigFile();
};

#endif
```

```
#ifndef CONFIG_H
#define CONFIG_H

#include <sstream>

class Config
{
protected:
    Config() {}
    virtual std::string readString(const std::string& key) const = 0;

public:
    virtual ~Config() {}

    template <typename T>
    T read(const std::string& key) const
    {
        std::istringstream iss(readString(key));
        T ret;
        iss >> ret;

        return ret;
    }
};

#endif
```

```
/* Seteos Ventana */
#define ANCHO_VENTANA 1024
#define ALTO_VENTANA 600

/* Elementos */
#define CANTIDAD_ELEMENTOS 8

#define ELEMENTO_A_BORRAR -2
#define NINGUN_ELEMENTO -1
#define MASA 0
#define PUNTO_FIJO 1
#define RUEDA 2
#define BARRA 3
#define PLATAFORMA 4
#define SOGA 5
#define CINTA 6
#define COHETE 7
#define BORRAR_TODO 8
#define BORRAR_ELEMENTO 9
#define CIRCULAR 10
#define LONGITUDINAL 11
#define ENLAZABLE 12
#define PUNTO_ENLACE 13
#define ZONA_LLEGADA 16
#define VOID 17

/* Texturas para paneles */
#define TEXTURA_DEFECTO 0
#define TEXTURA_MOUSEDOWN 1
#define TEXTURA_MOUSEOVER 2
#define TEXTURA_INHABILITADO 3
/* Texturas para el mapa */
#define TEXTURA_MASA 4
#define TEXTURA_PUNTO_FIJO 5
#define TEXTURA_RUEDA 6
#define TEXTURA_BARRA 7
#define TEXTURA_PLATAFORMA 8
#define TEXTURA_SOYA 9
#define TEXTURA_CINTA 10
#define TEXTURA_COHETE 11
#define PLAY 12
#define SALIR 13
#define SMS 14
#define TEXTURA_ZONA_LLEGADA 15

/* Seteos graficos */
#define POLY_SIDES 12

/* Tipos de seleccion de elemento */
#define ELEMENTO_NO_SELECCIONADO 0
#define ELEMENTO_SELECCIONADO POR_EXTREM01 1
#define ELEMENTO_SELECCIONADO POR_EXTREM02 2
#define ELEMENTO_SELECCIONADO POR_CENTRO 3
#define MARGEN_DE_AGARRE 20
#define MARGEN_DE_AGARRE_ANCHO 5
#define LONGITUD_MINIMA 20
#define LONGITUD_MAXIMA 100
#define LONGITUD_COHETE 100
#define PI 3.1415926535897932384626433832795
#define CTE_A 2000
#define CTE_B 1
#define CTE_C 4000
#define CTE_D 1
```

```
#ifndef CONSTSMS
#define CONSTSMS

/* Seteos Tiempo */
#define TIEMPOGANAR 4000.0
#define TIME_C 1000.0

#define CLOCK_X -1.6
#define CLOCK_Y -0.9

/* Definiciones Físicas */
#define RADIOMASA 2.0/30.0
    const double LONA_K = 40000;
    const double METAL_K = 50000;
    const double METAL_WIDTH=0.07;
    const double RUEDA_K = 10000;
    const long ROCKET_K=80000;
    const double ROCKET_WIDTH=0.07;
    const double ROPE_K = 10000;
    const double BALL_K = 10000;

/*Definiciones Gráficas*/
#define tX 300
#define tY 300
#define WIDTH 1000
#define HEIGHT 600

#endif
```

```
#ifndef COORDENADA_H_
#define COORDENADA_H_
#include <string>
#include "Angulo.h"
typedef unsigned long ulong;

class Coordenada {
public:
    /* Atributo publico X */
    long x;
    /* Atributo publico Y */
    long y;
    /* Atributo publico modulo */
    ulong modulo;
    /* Atributo publico angulo */
    long angulo;
    /* Contructor vacio */
    Coordenada();
    /* Contructor con parametros x e y */
    Coordenada(long x, long y);
    /* Contructor con parametros modulo y angulo */
    Coordenada(ulong modulo, long angulo);
    /* Destructor virtual */
    virtual ~Coordenada();
    /* Se modifica en delta */
    void modificarDelta(Cordenada);
    /* Mueve a otra coordenada */
    void modificar(Cordenada);
    /* Setea X, modifica angulo y modulo */
    void setX(long x);
    /* Setea Y, modifica angulo y modulo */
    void setY(long y);
    /* Setea angulo, modifica X e Y */
    void setAngulo(long angulo);
    /* Setea modulo, modifica X e Y */
    void setModulo(ulong modulo);
    /* Imprime por pantalla el valor */
    void imprimir() const;
    /* Operador Suma */
    Coordenada operator+(const Coordenada) const;
    /* Operador Resta */
    Coordenada operator-(const Coordenada) const;

private:
    void actualizarModulo();
    void actualizarAngulo();
    void actualizarX();
    void actualizarY();

};

#endif /*COORDENADA_H_*/
```

```
#include <time.h>
#include <ctime>
class Cronometro{
public:
    /*Resetea el cronometro y comienza a contar*/
    void iniciar();
    /*Detiene el cronometro y almacena el tiempo transcurrido
    desde que se inicio*/
    void detener();
    /*Devuelve el tiempo desde el inicio en ms.*/
    long getMilisegundos();
    /*Devuelve:
        0 si esta encendido
        1 si está parado*/
    int estaParado();

private:
    int parado;
    clock_t final, comienzo;
};
```

```
#ifndef ELEMENTO_H
#define ELEMENTO_H
#include <vector>

class Element
{
public:
    Element(unsigned f, unsigned l);
    /*Devuelve la primera masa del elemento */
    unsigned getFirst();
    /*Devuelve la ultima masa que constituye
    el elemento */
    unsigned getLast();
    /*Vector que contiene los numeros de
    las masas que componen el elemento*/
    std::vector<int> masas;
private:
    unsigned first;
    unsigned last;
};

#endif
```

```
#ifndef ELEMENTMANAGER_H
#define ELEMENTMANAGER_H

#include <cmath>
#include <iostream>
#include <vector>
#include <map>
#include <utility>
#include "element.h"
#include "zonaLlegada.h"

class ElementManager
{
public:
    ElementManager();
    ~ElementManager();
    /*Agrega una rueda al sistema*/
    void addRope(int first, int last);
    /*Agrega una cinta de lona al sistema*/
    void addLona(int first, int last);
    /*Agrega una barra de metal al sistema*/
    void addMetalBar(int first, int last);
    /*Agrega una plataforma metalica al sistema*/
    void addPlataforma(int first, int last);
    /*Agrega un cohete al sistema*/
    void addCohete(int first, int last);
    /*Agrega una rueda al sistema*/
    void addRueda(int first, int last);
    /*Agrega la masa principal al sistema*/
    void addMainBall(int first,int last);
    /*Agrega un punto fijo*/
    void addPuntoFijo(int first,int last);
    /*Agrega una zona de llegada*/
    void addZonaLlegada(double x,double y,double width,double heighth);
    /*Devuelve el vector contenido las sogas*/
    std::vector<Element*> getRopes();
    /*Devuelve el vector contenido las ruedas*/
    std::vector<Element*> getRuedas();
    /*Devuelve el vector contenido las barras*/
    std::vector<Element*> getMetalBars();
    /*Devuelve el vector contenido las plataformas*/
    std::vector<Element*> getPlataformas();
    /*Devuelve el vector contenido las cintas de lona*/
    std::vector<Element*> getLonas();
    /*Devuelve el vector contenido los cohetes*/
    std::vector<Element*> getCohetes();
    /*Devuelve el vector contenido las masas*/
    std::vector<Element*> getMainBalls();
    /*Devuelve el vector contenido los puntos fijos*/
    std::vector<Element*> getPuntosFijos();
    /*Devuelve el vector contenido las zonas de llegada*/
    std::vector<ZonaLlegada*> getZonasLlegada();
    /*El vector que contiene todos los vectores
     * de elementos (cohetes, ruedas, barras, etc.) */
    std::vector< std::vector<Element*>*> vectoresElementos;

private:
    std::vector<Element*> ropes;
    std::vector<Element*> cohetes;
    std::vector<Element*> ruedas;
    std::vector<Element*> plataformas;
    std::vector<Element*> metalBars;
    std::vector<Element*> mainBalls;
    std::vector<Element*> lonas;
    std::vector<Element*> puntosFijos;
    std::vector<ZonaLlegada*> zonasLlegada;

};

#endif
```

```
#ifndef ELEMENTO_H_
#define ELEMENTO_H_

#include <iostream>
#include "Coordenada.h"
#include <sstream>
#include "constantes.h"

typedef unsigned long ulong;

class Elemento {
private:
    ulong magnitud; /* Magnitud del elemento. Se adecua al tipo de elemento */
    long angulo; /* Angulo del elemento */
    Coordenada posicion; /* Posicion del punto representativo del elemento */
    bool bloqueado; /* Indica si el elemento puede ser modificado */
    std::string nombre; /* Nombre y jerarquia del elemento */
    bool mostrarEnlaces; /* Si debe mostrar sus enlaces */

public:
    /* Contructor vacio */
    Elemento();
    /* Contructor con parametros */
    Elemento(const std::string & nombre, Coordenada posicion,
             const ulong magnitud, long angulo);
    /* Contructor virtual */
    virtual ~Elemento();

    /* Redimensiona la longitud en deltaUnidades. */
    void setMagnitud(const long valor);
    /* Devuelve el valor de la magnitud del elemento */
    ulong getMagnitud() const;
    /* Da un valor al angulo */
    virtual void setAngulo(const long);
    /* Devuelve el angulo */
    long getAngulo() const;
    /* Guarda la posicion en la que se encuentra el Elemento */
    virtual void setPosicion(const Coordenada nuevaPosicion);
    /* Devuelve la posicion en la que se encuentra el Elemento */
    Coordenada getPosicion() const;
    /* Devuelve el extremo opuesto calculado por trigonometria */
    virtual Coordenada getExtremoOpuesto() const;
    /* Bloquea al elemento: no se puede mover, redimensionar ni borrar */
    void bloquear();
    /* Desbloquea al elemento: no se puede mover, redimensionar ni borrar */
    void desbloquear();
    /* True si no es movible, rotable ni redimensionable */
    bool estaBloqueado() const;
    /* Devuelve el nombre del elemento en un string */
    std::string getNombre() const;
    /* Dice si es un determinado elemento, pregunta a getNombre */
    bool es(int const nro_elemento) const;
    /* Devuelve el tipo de elemento */
    int getTipo() const;
    /* Devuelve true si la Coordenada pertenece al Elemento */
    virtual int seleccionar(Coordenada) const = 0;
    /* Imprime por stdout la informacion del elemento */
    virtual void imprimir();
    /* Dice si deben aparecer los puntos de enlace */
    bool getMostrarPuntosDeEnlace() const;
    /* Asigna si deben aparecer los puntos de enlace */
    void setMostrarPuntosDeEnlace(const bool);

};

#endif /*ELEMENTO_H_*/
```

```
#ifndef ELEMENTOCIRCULAR_H_
#define ELEMENTOCIRCULAR_H_

#include "Elemento.h"
#include <string>

class ElementoCircular : public Elemento {

public:
    ElementoCircular();
    ElementoCircular(const std::string & nombre, Coordenada posicion,
                    const ulong magnitud, long angulo);
    virtual ~ElementoCircular();

    /* Devuelve true si la Coordenada pertenece al Elemento */
    virtual int seleccionar(Coordenada) const;
};

#endif /*ELEMENTOCIRCULAR_H_*/
```

```
#ifndef ELEMENTOENLAZABLE_H_
#define ELEMENTOENLAZABLE_H_

#include "ElementoCircular.h"
#include "Elemento.h"
#include "Coordenada.h"
#include <list>

class ElementoEnlazable : public ElementoCircular
{
private:
    Elemento * poseedor;
    std::list < ElementoEnlazable* > _enlaces;

public:
    ElementoEnlazable(const std::string & nombre,
                      Coordenada posicion, const ulong magnitud, long angulo, Elemento * poseedor);
    virtual ~ElementoEnlazable();

    /* Devuelve el Elemento que lo contiene */
    Elemento * getPoseedor() const;
    /* Devuelve la lista de enlazables que tiene asociados */
    std::list <ElementoEnlazable*> getListaEnlaces() const;
    /* Vacia los enlaces y los libera */
    void eliminarEnlaces();
    /* Agrega un enlace a la lista verificando y devuelve si pudo
     hacerlo (verifica que su posicion sea igual que la propia)
     */
    bool conectarA(ElementoEnlazable* );
    bool enlazado;

};

#endif /*ELEMENTOENLAZABLE_H_*/
```

```
#ifndef ELEMENTOLONGITUDINAL_H_
#define ELEMENTOLONGITUDINAL_H_

#include "Elemento.h"
#include <string>
#include <list>
#include "PuntoDeEnlace.h"

#define LONGITUD_DE_ENLACE_MINIMA 50

class ElementoLongitudinal : public Elemento {

private:
    std::list <PuntoDeEnlace*> _enlaces;
    Coordenada extremoOpuesto;

protected:
    ulong ancho;

public:
    ElementoLongitudinal();
    ElementoLongitudinal(const std::string & nombre, Coordenada posicion,
                         const ulong magnitud, long angulo, ulong ancho);
    virtual ~ElementoLongitudinal();

    /* Agrega a la lista los puntos de enlaces en funcion de la longitud
     * determinando su posicion */
    void actualizarPuntosDeEnlace(int modo);
    /* Imprime los enlaces */
    void imprimirEnlaces();
    /* Devuelve puntero a lista de enlaces */
    std::list<PuntoDeEnlace*> getListadeEnlaces();
    /* Elimina los enlaces */
    void eliminarEnlaces();
    /* Devuelve ancho */
    ulong getAncho() const;
    /* Setea el extremo opuesto y actualiza el modulo y angulo */
    void setExtremoOpuesto(const Coordenada);
    /* Devuelve el extremo Opuesto */
    Coordenada getExtremoOpuesto() const;
    /* Devuelve true si la Coordenada pertenece al Elemento */
    virtual int seleccionar(Coordenada) const;
    /* Redefino setPosicion. Actualizo extremoOpuesto */
    void setPosicion(const Coordenada nuevaPosicion);
    /* Redefino setAngulo. Actualizo extremoOpuesto */
    void setAngulo(const long angulo);

};

#endif /*ELEMENTOLONGITUDINAL_H_*/
```

```
#ifndef FABRICADEELEMENTOS_H_
#define FABRICADEELEMENTOS_H_

#include "Coordenada.h"
#include "Elemento.h"
#include "Masa.h"
#include "Rueda.h"
#include "PuntoFijo.h"
#include "Barra.h"
#include "Soga.h"
#include "Plataforma.h"
#include "Cinta.h"
#include "Cohete.h"
#include "constantes.h"
#include "Llegada.h"

class FabricaDeElementos
{
public:
    FabricaDeElementos();
    virtual ~FabricaDeElementos();
    Elemento * crear (int nroElemento, Coordenada posicion);
};

#endif /*FABRICADEELEMENTOS_H_*/
```

```
// Coded by Mariano M. Chouza (http://www.chouza.com.ar) replacing a version
// present in "Sol's 2d gl basecode 2.0" (http://www.iki.fi/sol/)

#ifndef FILEUTILS_H
#define FILEUTILS_H

#include <fstream>
#include <SDL.h>

class BinFile
{
    std::fstream s_;
public:
    BinFile(const char *filename);
    virtual ~BinFile();
    char readbyte();
    int16_t readword();
    int32_t readdword();
    void readbytes(char* buffer, size_t num);
    size_t tell();
    void seek(size_t pos);
};

#endif
```

```
#ifndef __HILO_CRON_H__
#define __HILO_CRON_H__

#include "common_SocketCliente.h"
#include "common_Thread.h"
#include <iostream>
#include "cronometro.h"

class HiloCron: public Thread
{
private:
    Cronometro* cronometro;

public:
    /* Constructor */
    HiloCron();

    /* Destructor */
    virtual ~HiloCron();

    long getMilisegundos();

    /* Función que ejecuta el hilo. */
    void run();
};

#endif
```

```
#ifndef INTERFAZEDICION_H_
#define INTERFAZEDICION_H_


#include <iostream>
#include "Angulo.h"
#include <string>
#include "Mapa.h"
#include "Masa.h"
#include "Barra.h"
#include "Ventana.h"
#include "PanelContenedor.h"
#include "Boton.h"
#include "InterfazMapa.h"
#include "constantes.h"
#include "Elemento.h"

/* Eventos sobre el mapa */
void mapa_mouseDown(Coordenada punto, InterfazMapa *);
void mapa_mouseOver(Coordenada punto, InterfazMapa *);
void mapa_mouseUp(Coordenada punto, InterfazMapa *);
/* Eventos de la botonera */
void botonMasa_mouseDown(Coordenada cord, Boton *);
void botonRueda_mouseDown(Coordenada cord, Boton *);
void botonPuntoFijo_mouseDown(Coordenada cord, Boton * boton);
void botonBarra_mouseDown(Coordenada cord, Boton *);
void botonPlataforma_mouseDown(Coordenada cord, Boton * boton);
void botonSoga_mouseDown(Coordenada cord, Boton * boton);
void botonCinta_mouseDown(Coordenada cord, Boton * boton);
void botonZonaLlegada_mouseDown(Coordenada cord, Boton * boton);
void botonCohete_mouseDown(Coordenada cord, Boton * boton);
void botonBorrarTodo_mouseDown(Coordenada cord, Boton * boton);
void botonBorrarElemento_mouseDown(Coordenada cord, Boton * boton);
void botonPlay_mouseDown(Coordenada cord, Boton * boton);
void botonSalir_mouseDown(Coordenada cord, Boton * boton);
void botonSms_mouseDown(Coordenada cord, Boton * boton);

class InterfazEdicion {

private:
    /* Referencia al modelo del mapa sobre el cual editara */
    Mapa * mapa;
    /* Agrega los botones */
    void agregarBotoneraInferior(Ventana *);

public:
    InterfazEdicion(Mapa*);
    ~InterfazEdicion();
    void comenzar(ChatDoble* chat);

};

#endif /*INTERFAZEDICION_H_*/
```

```
#ifndef INTERFAZMAPA_H_
#define INTERFAZMAPA_H_

#include "Coordenada.h"
#include "PanelContenedor.h"
#include "Mapa.h"
#include <string>
#include "ElementoCircular.h"
#include "ElementoLongitudinal.h"
#include "Llegada.h"
class InterfazMapa : public PanelContenedor
{
private:
    /* Visibilidad del modelo mapa */
    Mapa * mapa;
    /* Rutinas del mouse */
    void (*pf_rutinaMouseDown)(Coordenada, InterfazMapa *);
    void (*pf_rutinaMouseUp)(Coordenada, InterfazMapa *);
    void (*pf_rutinaMouseOver)(Coordenada, InterfazMapa *);

    /* Metodos para imprimir cada tipo de elemento */
    void imprimirElementoCircular(ElementoCircular * circular);
    void imprimirElementoLongitudinal(ElementoLongitudinal * longitudinal);
    void imprimirZonaLlegada(Llegada * llegada);

public:
    /* Constructores y destructores, con y sin zoom de vision */
    InterfazMapa(const std::string & rutaImgDefecto, const std::string & rutaImgClickDown, Mapa * mapa);
    InterfazMapa(const std::string & rutaImgDefecto, const std::string & rutaImgClickDown, Mapa * mapa, double zoom);
    virtual ~InterfazMapa();

    /* Redefinición de llamadas a eventos */
    void evento_MouseDown(Coordenada punto);
    void evento_MouseUp(Coordenada punto);
    void evento_MouseOver(Coordenada punto);

    /* setters para las rutinas de cada evento */
    void setRutinaMouseDown(void (*rutina)(Coordenada, InterfazMapa *));
    void setRutinaMouseUp(void (*rutina)(Coordenada, InterfazMapa *));
    void setRutinaMouseOver(void (*rutina)(Coordenada, InterfazMapa *));
    Coordenada vectorDiferencia;
    /* getter y setter del mapa */
    Mapa * getMapa() const;
    void setMapa(Mapa* mapa); //destruye el viejo
    /* Dibuja el mapa */
    void imprimir();

};

#endif
```

```
#ifndef LLEGADA_H_
#define LLEGADA_H_

#include <iostream>
#include "Coordenada.h"
#include "constantes.h"
#include "Elemento.h"

class Llegada : public Elemento {

public:
    Llegada(Coordenada);
    int seleccionar(Coordenada) const;

};

#endif /*LLEGADA_H_*/
```

```
#ifndef TEST_SMS_H
#define TEST_SMS_H

#include "spring_mass_system.h"

/*Instancia el modelo en forma de masas y resortes
basandose en el mapa recibido de la etapa de
edicion*/
void load(SpringMassSystem& sms);

#endif
```

```
#ifndef MAPA_H_
#define MAPA_H_
#include "Elemento.h"
#include <list>
#include "FabricaDeElementos.h"
#include "constantes.h"
typedef unsigned long ulong;

class Mapa {
private:
    /* Nombre del mapa. Sirve para buscar el archivo de imagen de fondo */
    std::string nombre;
    /* Ancho y alto del mapa */
    ulong ancho;
    ulong alto;
    /* Lista de elementos contenidos en mapa */
    std::list<Elemento*> _elementos;
    /* Es el elemento seleccionado. Todos los metodos trabajaran sobre el */
    Elemento * elemento;
    /* Es el elemento que se creará al llamar agregarElemento() */
    int nroElementoParaCrear;
    /* Indica por donde se a seleccionado el elemento */
    int tipoSeleccion;
    /* Vector de elementos habilitados */
    bool elementosHabilitados [CANTIDAD_ELEMENTOS];
    /* Indica los precios de los elementos de este mapa en particular */
    unsigned int precio[CANTIDAD_ELEMENTOS];
    /* Indica la plata que este mapa provee */
    int plata;
    /* Indica el puntaje */
    int puntos;
    int tiempoCohete; // milisegundos. 0 es tiempo infinito
    long tiempoResolucion; // milisegundos de tiempo de resolucion del mapa
public:
    /* Contructor con valores por default */
    Mapa();
    /* Contructor con parametros */
    Mapa(ulong ancho, ulong alto);
    /* Destructor virtual */
    virtual ~Mapa();
    /* Devuelve ancho */
    ulong getAncho() const;
    /* Devuelve alto */
    ulong getAlto() const;
    /* Setea el elemento que la fabrica contruira cuando se lo pida */
    void setNroElementoParaCrear(int);
    /* Devuelve el elemento a crear */
    int getNroElementoParaCrear() const;
    /* Agrega elemento al mapa. Devuelve true si fue posible */
    bool agregarElemento(Coordenada posicion);
    /* Selecciona el primer elemento que encuentre en esa coordenada.
     * Devuelve False en caso de no haber ninguno
     */
    int seleccionarElemento(Coordenada);
    /* Informa si hay algun elemento seleccionado */
    bool hayElementoSeleccionado() const;
    /* Devuelve por donde se ha seleccionado el elemento */
    int getTipoSeleccion() const;
    /* setea el tipo de seleccion de forma forzada */
    void setTipoSeleccion(const int tipo);
    /* Devuelve puntero constante al Elemento seleccionado */
    const Elemento * getElementoSeleccionado() const;
    /* Devuelve cantidad de elementos en el mapa */
    size_t getCantidadElementos();
    /* Devuelve true si existe punto esta en el entorno
     * de algun punto de enlace.
     * Retorna en puntoExacto el centro del punto de enlace
     */
    bool entornoDePuntoDeEnlace(Coordenada punto, Coordenada * puntoExacto);
    /* Setea el extremo opuesto del elemento longitudinal seleccionado */
    void setExtremoOpuesto(const Coordenada punto);
    /* Devuelve el extremo opuesto del elemento longitudinal seleccionado */
}
```

```
Coordenada getExtremoO puesto() const;
/* Mueve el elemento a destino. Devuelve true si fue posible */
bool moverElementoA(Cordenada posicion);
/* Rota el elemento a destino. Devuelve true si fue posible */
bool rotarElementoA(long angulo);
/* Agranda el elemento a longitud. Devuelve true si fue posible */
bool agrandarElementoA(long longitud);
/* Actualiza modulo y angulo en funcion del extremoO puesto.
 * Mantiene posicion constante.
 * Retorna la coordenada corregida.
 */
Coordenada actualizarModuloAngulo(Cordenada extremoO puesto);
/* Actualiza solo el angulo manteniendo LONGITUD_MINIMA */
Coordenada actualizarAngulo(Cordenada extremoO puesto);
/* Actualiza solo el angulo manteniendo LONGITUD_MAXIMA */
Coordenada actualizarAngulo2(Cordenada click);
/* Elimina el elemento y extrae de la lista*/
void eliminarElemento();
/* Elimina todos y vacia la lista */
void eliminarTodosLosElementos();
/* Imprime todos los elementos */
void imprimir();
/* Devuelve la lista de elementos */
std::list <Elemento*> getListadoElementos() const;
/* Actualiza los puntos de enlace de los elementos longitudinales */
void actualizarEnlaces();
/* Si el elemento seleccionado debera ser eliminado */
bool elementoABorrar;
/* Filtra los enlaces en todo el mapa y los del elementos longitudinales */
std::list <ElementoEnlazable*> getTodosLosEnlaces();
/* Avisa al Elemento Seleccionado si mostrar sus puntos de enlace */
void setMostrarPuntosDeEnlaceElemento(const bool);
/* Bloquea todos los elementos del mapa */
void bloquearElementos();
/* Desbloquea todos los elementos del mapa */
void desbloquearElementos();
/* Deshabilita un elemento */
void deshabilitarElemento(const int nroElemento);
/* Habilita un elemento */
void habilitarElemento(const int nroElemento);
/* Devuelve si el elemento esta habilitado */
bool elementoHabilitado(const int nroElemento) const;
/* Muestra por consola los enlaces */
void imprimirTodosLosEnlaces();

/* Otros getters y setter de menor importancia */
void setNombre(const std::string&);
std::string getNombre() const;
void setPlata(const int);
int getPlata() const;
void setPuntos(const int);
int getPuntos() const;
void setPrecio(int nroElemento, int precio);
int getPrecio(int nroElemento) const;
int getTiempoCohete() const; // milisegundos
void setTiempoCohete(int tiempo); // milisegundos
void setTiempoResolucion(const long);
long getTiempoResolucion() const;
int getSumaDePrecios() const;

};

#endif /*MAPA_H_*/
```

```
#ifndef MASA_H_
#define MASA_H_

#include "ElementoCircular.h"

class Masa : public ElementoCircular {
public:
    /* Constructor por defecto */
    Masa();
    /* Constructor con parametros */
    Masa(Coordenada posicion, ulong magnitud);
    /* Destructor virtual */
    virtual ~Masa();

};

#endif /*MASA_H_*/
```

```
#ifndef PANELCONTENEDOR_H_
#define PANELCONTENEDOR_H_
#include "SDL.h"
#include "SDL_image.h"
#include <SDL_opengl.h>
#include "Coordenada.h"
#include <list>
#include <map>
#include <string>
#include <math.h>
#include "Ventana.h"
#include "constantes.h"

class Ventana;

typedef unsigned long ulong;

class PanelContenedor
{
private:
    /* ancho pantalla */
    ulong ancho;
    /* alto pantalla */
    ulong alto;
    /* Posicion del panel en pantalla */
    Coordenada posicion;
    /* numero de textura que se imprime actualmente */
    int texturaActual;
    /* Referencia a la pantalla */
    SDL_Surface * pantalla;
    /* Visibilidad de la textura */
    bool visible;
    /* Lista de paneles contenidos */
    std::list<PanelContenedor*> _paneles;
    /* Indica si fue clickeado */
    bool clicked;
    /* Indica si tiene el mouse arriba */
    bool mouseOver;
    /* Nombre en la jerarquia */
    std::string nombre;
    /* Id de panel */
    int id;

    ulong getAncho(const std::string & rutaImg);
    ulong getAlto(const std::string & rutaImg);

protected:
    /* mapa de texturas */
    std::map<int, GLuint> _texturas;
    /* Referencia de la ventana contenedora */
    Ventana * ventana;
    /* Carga una textura al map */
    void loadTexture(const char * path, int tipoTextura);
    /* Indica si esta habilitado */
    bool habilitado;

public:
    /* Constructores y destructor */
    PanelContenedor(const std::string & nombre, const std::string & rutaImgDefecto, const std::string & rutaImgClickDown, const std::string & rutaImgMouseOver);
    PanelContenedor(const std::string & nombre, const std::string & rutaImgDefecto, const std::string & rutaImgClickDown, const std::string & rutaImgMouseOver, double zoom);
    virtual ~PanelContenedor();
    /* Imprime en pantalla el panel y subpaneles */
    virtual void imprimir();
    /* Imprime la textura actual */
    void imprimirImagenDeFondo();
    /* Agregan paneles a la lista */
    void agregarPanel(PanelContenedor *, Coordenada);
    void agregarPanel(PanelContenedor * panel, int lugar);
```

```
/* Utilitarios para la conversion de cordenadas SDL a OpenGL */
double cambioX(long x);
double cambioY(long y);
/* Devuelve true si la coordenada pertenece al panel */
bool pertenece(Cordenada);
/* Ejecuta evento o bien lo ejecuta en un subpanel */
bool seEjecutaEventoEn(const Cordenada punto, int evento);

//SETTERS Y GETTERS
void setPantalla (SDL_Surface * pantalla);
void setVentana(Ventana *);
void setPosicion(const Cordenada posicion); //la setea el agregador de panel
Cordenada getPosicion() const;
void setImagenDeFondo(int);
void setAltura(const ulong);
void setAncho(const ulong);
ulong getAltura() const;
ulong getAncho() const;
bool is_clicked() const;
bool is_over() const;
std::string getNombre() const;
int getId() const;
void setId(const int);
void setMouseOver(const bool b);
bool estaHabilitado() const;
void setHabilitar(const bool);

//EVENTOS
void virtual evento_MouseDown(Cordenada punto);
void virtual evento_MouseOver(Cordenada punto);
void virtual evento_MouseUp(Cordenada punto);

//VARIABLES DE CONVERSION DE SDL A OPENGL
double TX;
double TY;
};

#endif /*PANELCONTENEDOR_H_*/
```

```
#ifndef __PARSER_XML_H__
#define __PARSER_XML_H__

#include "Mapa.h"
#include "Elemento.h"
#include <string>
#include <libxml/xmlreader.h>

class ParserXML
{
public:
    /* Devuelve un mapa en base a un XML levantado de un archivo */
    Mapa* obtenerMapaArchivo (const std::string& rutaEntrada) const;

    /* Devuelve un mapa en base a un XML que está en memoria y se pasa
     * por parámetro */
    Mapa* obtenerMapaMemoria (const std::string& xml) const;

    /* Se genera un archivo XML en base al mapa dado */
    void obtenerXMLArchivo (const std::string& rutaSalida, const Mapa* mapa) const;

    /* Se genera un XML en Memoria en base a un mapa dado */
    void obtenerXMLMemoria (std::string& xml, const Mapa* mapa) const;

private:
    /* Analiza el contenido de un nodo y en base al nombre del tag invoca al método
     * correspondiente */
    void analizarNodo(Mapa* mapa, xmlTextReaderPtr reader) const;

    /* Convierte un entero a un string */
    void enteroAString(std::string& cadena, ulong entero) const;

    /* Analiza un elemento en su totalidad y lo agrega al mapa */
    void procesarElemento(Mapa* mapa, xmlTextReaderPtr reader) const;

    /* Lee los costos de cada elemento */
    void procesarCostos(Mapa* mapa, xmlTextReaderPtr reader) const;

    /* Analiza qué elementos debe habilitar y cuáles no */
    void procesarHabilitados(Mapa* mapa, xmlTextReaderPtr reader) const;

    /* Imprime qué botones están habilitados y cuales no en el mapa */
    void imprimirHabilitados(std::string& xml, const Mapa* mapa) const;

    /* Imprime todos los elementos del mapa */
    void imprimirElementos(std::string& xml, const Mapa* mapa) const;

    /* Obtiene un mapa a partir del lector dado */
    Mapa* obtenerMapa (xmlTextReaderPtr reader) const;

    /* Imprime el tiempo del cohete */
    void imprimirTiempo(std::string& xml, const Mapa* mapa) const;

    /* Imprime los costos de cada elemento */
    void imprimirCostes(std::string& xml, const Mapa* mapa) const;

    /* Determina el número de elemento según el tipo dado */
    int nroElemento(const std::string& tipo) const;

};

#endif
```

```
#ifndef PLATAFORMA_H_
#define PLATAFORMA_H_

#include "ElementoLongitudinal.h"

class Plataforma : public ElementoLongitudinal
{
public:
    Plataforma();
    Plataforma(Coordenada posicion, ulong magnitud, long angulo, long ancho);
    virtual ~Plataforma();
};

#endif /*PLATAFORMA_H_*/
```

```
#ifndef PUNTODEENLACE_H_
#define PUNTODEENLACE_H_

#include "ElementoEnlazable.h"

class PuntoDeEnlace : public ElementoEnlazable
{
public:
    PuntoDeEnlace();
    PuntoDeEnlace(Coordenada posicion, Elemento * poseedor);
    virtual ~PuntoDeEnlace();
};

#endif /*PUNTODEENLACE_H_*/
```

```
#ifndef PUNTOFIJO_H_
#define PUNTOFIJO_H_

#include "ElementoEnlazable.h"

class PuntoFijo : public ElementoEnlazable
{
public:
    PuntoFijo();
    PuntoFijo(Coordenada posicion, ulong magnitud);
    virtual ~PuntoFijo();
};

#endif /*PUNTOFIJO_H_*/
```

```
#ifndef __RELOJ_H__
#define __RELOJ_H__

#include <string>
#include <map>
#include <SDL.h>
#include "SDL_image.h"
#include <SDL_opengl.h>

class Reloj
{
    private:
        std::map<char,GLuint> texturas;
        void imprimirCaracter(char letra, float x, float y);
        void loadTexture(const char * path, char letra);

    public:
        void imprimirReloj(const std::string& reloj, float x, float y);
        void cargarTexturas();

};

#endif
```

```
#ifndef RUEDA_H_
#define RUEDA_H_

#include "ElementoCircular.h"
#include <list>
#include "PuntoDeEnlace.h"

class Rueda : public ElementoCircular {
private:
    std::list<PuntoDeEnlace*> _enlaces;

public:
    /* Constructor por defecto */
    Rueda();
    /* Constructor con parametros */
    Rueda(Coordenada posicion, ulong magnitud);
    /* Destructor virtual */
    virtual ~Rueda();
    /* Ubica los enlaces */
    void actualizarPuntosDeEnlace();
    /* Imprime los enlaces */
    void imprimirEnlaces();
    /* Devuelve puntero a lista de enlaces */
    std::list<PuntoDeEnlace*> getListaEnlaces();
    /* Elimina los enlaces */
    void eliminarEnlaces();

};

#endif /*RUEDA_H_*/
```

```
#ifndef SMS_APP_H
#define SMS_APP_H

#include "config_file.h"
#include "spring_mass_system.h"
#include "video.h"
#include "hiloCron.h"
#include <string>
#include "Reloj.h"
#include <SDL.h>

class SMSApp
{
    /*Muestra el tiempo de simulación en pantalla*/
    Reloj reloj;
    /*Carga el archivo de configuracion*/
    ConfigFile config_;
    /*Contiene información del sistema de masas y resortes*/
    SpringMassSystem sms_;
    /*Carga y maneja texturas y configuraciones gráficas*/
    Video video_;
    uint32_t ticks_;
    int physMult_;
    /*Dibuja la situación actual del SMR*/
    void drawFrame(std::string & time);
    /*Devuelve true si se cumple el objetivo del juego*/
    bool ganaNivel();
    /*Verifica la posición de la masa principal comparandola con
     las areas de llegada*/
    bool estaEnLlegada();
    bool estabaEnLlegada;
    /*Thread que controla el tiempo de simulación*/
    HiloCron* tiempoLlegada;
    HiloCron* cronometro;
public:
    SMSApp(int argc, Mapa* mapa);
    virtual ~SMSApp();
    double run();
    bool sms;
};

#endif
```

```
#ifndef SOGA_H_
#define SOGA_H_

#include "ElementoLongitudinal.h"

class Soga : public ElementoLongitudinal
{
public:
    Soga();
    Soga(Coordenada posicion, ulong magnitud, long angulo, long ancho);
    virtual ~Soga();
};

#endif /*SOGA_H_*/
```

```
#ifndef SPRING_MASS_SYSTEM_H
#define SPRING_MASS_SYSTEM_H
#include <algorithm>
#include "config.h"
#include "vector2.h"
#include "hiloCron.h"
#include "elementManager.h"
#include <utility>
#include <vector>
#include <list>
#include "Mapa.h"

class SpringMassSystem;

class MassProxy
{
    friend class SpringMassSystem;

    SpringMassSystem& SMS_;
    size_t index_;

    MassProxy(SpringMassSystem& sms, size_t index);

public:
    /*Devuelve el valor de una masa determinada*/
    double getInvMass() const;

    /*Devuelve la posicion de una masa determinada*/
    const Vector2<>& getPos() const;

    /*Devuelve la posicion de la masa*/
    const Vector2<>& getVel() const;

    /*Setea la velocidad de dicha masa*/
    void setVel(const Vector2<>& v);
};

class SpringProxy
{
    friend class SpringMassSystem;

    SpringMassSystem& SMS_;
    size_t index_;

    SpringProxy(SpringMassSystem& sms, size_t index);

public:
    /*Devuleve la posición de la masa en
     *la que comienza el resorte*/
    const Vector2<>& getStartPos() const;

    /*Devuelve la posicion de la masa hasta
     *donde llega el resorte*/
    const Vector2<>& getEndPos() const;

    /*Devuelve la elongacion del resorte
     *en el momento en que se llama*/
    double getRelElongation() const;

    /*Devuelve la constante del resorte*/
    double getK() const;
};

struct SMSStaticState
{
    // Masses data
    std::vector<double> invMassVec;

    // Springs data
    /*Se alamcenan los valores estaticos del sistema:
     . masas que une cada resorte
```

```
. constantes de los resortes
. elongaciones iniciales de los resortes
. resistencia de los resortes
*/
std::vector<std::pair<int, int> > springEndPointsVec;
std::vector<double> springKsVec;
std::vector<double> springNormalLengthsVec;
std::vector<double> springInvStrength;

// Gravity
Vector2<> g;

// Viscous damping
double damping;

// Collidable masses info
/*Almacena las masas que colisionan y
sus respectivos radios de colision*/
std::vector<int> collMassesIndices;
std::vector<double> collMassesRadii;
};

class SpringMassSystem
{
    // System dynamic state (the positions are stored in the first half and
    // velocities in the other half)
    std::vector<Vector2<> > y_;

    // System static state (the part of the state that doesn't change)
    SMSstaticState x_;

    // Friends
    friend class MassProxy;
    friend class SpringProxy;

    ElementManager manager;
    Mapa* mapa;
    std::vector<int> enganches;
    long rocketTime;
    HiloCron* cronometro;
public:
    SpringMassSystem(const Config& config);

    /*Agrega una masa al sistema*/
    int addMass(double invMass, const Vector2<>& pos);

    /*Agrega una masa que colisiona al sistema*/
    int addCollMass(double invMass, const Vector2<>& pos, double radius);

    /*Agrega un resorte al sistema*/
    int addSpring(int startMass, int endMass, double k,
                  double relElongation = 0.0, double invStrength = 0.0);

    /*Fija una masa determinada*/
    void fixMass(int pos);

    /*Devuelven los proxys de masas y resortes
    Se obtienen resortes y masas específicos determinados
    por el parametro index
    */
    MassProxy getMassProxy(size_t index);
    SpringProxy getSpringProxy(size_t index);

    /*Devuelve la cantidad de masas y resortes*/
    size_t getNumMasses() const;
    size_t getNumSprings() const;

    double getPotentialEnergy() const;
    double getKineticEnergy() const;
    double getEnergy() const;

    void eulerStep(double dt);
```

```
void rk4Step(double dt);

void debugPrint(std::ostream& os) const;

/*Unifica las dos masas pasados por parametros
esta funcion se utiliza para los enganches*/
void unifyMass(int first,int second);

/*Realiza las verificaciones de proximidad
para unificar las masas*/
void checkRoutine();

/*Devuelve un puntero al ElementManager*/
ElementManager* getElementManager();

/*Devuelve el vector con las masas que son
enganches, con estos verifica la proximidad*/
std::vector<int> getVecEnganches();

/*agrega una masa al vector de enganches*/
void addEnganche(int enganche);

/*Alamacena una referencia al mapa editado
para instanciar el sistema en base a este*/
void setMapa(Mapa* mapa);

void setCronometro(HiloCron *cronometro);
/*Setea el tiempo de cohetes*/
void setRocketTime(long time);
/*Devuelve la referencia al mapa*/
Mapa* getMapa();
};

#endif
```

```
#ifndef TEXTURE_STORE_H
#define TEXTURE_STORE_H

#include <map>
#include <string>
#include <SDL_opengl.h>

class TextureStore
{
    typedef std::map<std::string, GLuint> TTexStore;
    static TTexStore texStore_;
public:
    static GLuint addTexture(const std::string& filename);
    static GLuint getTexture(const std::string& filename);
};

#endif
```

```
#ifndef VECTOR2_H
#define VECTOR2_H

#include <cmath>
#include <iostream>

template <typename Real = double>
struct Vector2
{
    Real x;
    Real y;

    Vector2() : x(0), y(0) {}
    Vector2(const Vector2& v) : x(v.x), y(v.y) {}
    Vector2(Real x, Real y) : x(x), y(y) {}

    Vector2& operator+=(const Vector2& v)
    {
        x += v.x;
        y += v.y;
        return *this;
    }

    Vector2& operator-=(const Vector2& v)
    {
        x -= v.x;
        y -= v.y;
        return *this;
    }

    Vector2& operator*=(Real f)
    {
        x *= f;
        y *= f;
        return *this;
    }

    Vector2& normalize()
    {
        return *this *= 1.0 / norm();
    }

    double dot(const Vector2& v) const
    {
        return x * v.x + y * v.y;
    }

    double sqNorm() const
    {
        return x * x + y * y;
    }

    double norm() const
    {
        return sqrt(x * x + y * y);
    }

    friend std::ostream& operator<<(std::ostream& os, const Vector2& v)
    {
        os << v.x << " " << v.y;
        return os;
    }

    friend std::istream& operator>>(std::istream& is, Vector2& v)
    {
        is >> v.x >> v.y;
        return is;
    }
};

#endif
```

```
#ifndef VENTANA_H_
#define VENTANA_H_

#include <SDL.h>
#include <SDL_image.h>
#include "PanelContenedor.h"
#include "Coordenada.h"
#include <list>
#include <SDL_opengl.h>
#include "constantes.h"
#include "client_ChatDoble.h"
#include "Reloj.h"
#include "constantes_sms.h"
#include <string>
#include "Mapa.h"
#include "hiloCron.h"

class PanelContenedor;

class Ventana
{
private:
    /* Pantalla grafica */
    SDL_Surface * pantalla;
    /* Dimensiones pantalla */
    int ancho;
    int alto;
    /* Lista de paneles contenidos */
    std::list<PanelContenedor *> _paneles;
    /* Corrige las coordenadas */
    void corregirCoordenadas(ulong *x, ulong *y);
    /* Reloj que cuenta el tiempo */
    Reloj reloj;
    /* Hilo del contador de tiempo */
    HiloCron * cronometro;
    /* Muestra el dinero */
    Reloj displayDinero;
    /* Puente de comunicación */
    ChatDoble* chat;
    /* Calcula puntaje mediante una formula */
    long calcularPuntaje(long a, long b, long c, long d, long tS, long tR, int sumaPrecios);
public:
    /* Constructor y destructor */
    Ventana(int ancho, int alto, ChatDoble* chat);
    virtual ~Ventana();
    /* Recibe eventos y dibuja */
    void ejecutar (void* contrincante, void* mapa);
    /* Dibuja todos los paneles, Reloje y Plata */
    void imprimir_pantalla(std::string & tiempo, std::string & money);
    /* Agrega paneles */
    void agregarPanel(PanelContenedor *, const Coordenada);
    void agregarPanel(PanelContenedor * panel, const int lugar);
    /* Ejecuta eventos de paneles o propios */
    bool seEjecutaEventoEn(const Coordenada, int evento);
    /* Levanta a los botones porque otro se clickeo */
    void desclickearBotones(int id_excepcion);

    /* Eventos del mouse */
    void evento_MouseDown(Coordenada punto);
    void evento_MouseOver(Coordenada punto);
    void evento_MouseUp(Coordenada punto);
    /* variables utilitarias */
    bool salir;
    bool sms;
    bool pideSimulacion;
    long getTiempoResolucion();

};

#endif /*VENTANA_H_*/
```

```
#ifndef VIDEO_H
#define VIDEO_H

#include "config.h"

#include "loader.h"
#include <cmath>
#include <fstream>
#include "SDL.h"
#include "SDL_image.h"
#include <SDL_opengl.h>
#include <sstream>
#include <iostream>
#include <vector>

class Video
{
public:
    Video(const Config& config, const char* pathFondo);

    GLuint fondo;
    const char* pathFondo;
    std::vector<GLuint> textures;
    void loadSurfaces();
    void loadSurface(const char* surf);
};

#endif
```

```
#ifndef ZONA_LLEGADA_H
#define ZONA_LLEGADA_H

class ZonaLlegada
{
public:
    ZonaLlegada(double x, double y , double width, double heighth);
    /*Devuelven las propiedades de la zona de llegada*/
    /*posicion*/
    double getX();
    double getY();
    /*dimensiones*/
    double getWidth();
    double getHeighth();
private:
    double x;
    double y;
    double width;
    double heighth;
};

#endif
```

```
#include "Angulo.h"
#include <math.h>
#include <iostream>
using namespace std;

// hay q hacer bien esta funcion
void Angulo::corregir(long* valor)
{
    long copia = *valor;

    if(fabs(*valor) >= 360.0)
    {
        long vueltas = copia / 360.0;
        copia = copia - (vueltas*360.0);
    }

    if(*valor < 0)
        *valor = 360.0 - copia;
    else
        *valor = copia;
}

double Angulo::aRadianes(long grados) {
    return (2.0*PI * (double)grados) / 360.0;
}

long Angulo::aGrados(double radianes) {
    return ((radianes * 360.0) / (2.0*PI));
}
```

```
#include "Barra.h"

using namespace std;

Barra::Barra() :
    ElementoLongitudinal("Longitudinal:Barra", Coordenada(long(10), long(10)), 25, 0, 10) {
}

Barra::Barra(Cordenada posicion, ulong magnitud, long angulo, long ancho) :
    ElementoLongitudinal("Longitudinal:Barra", posicion, magnitud, angulo, ancho) {
}

Barra::~Barra() {
```

```
#include "Boton.h"
#include <iostream>
#include "constantes.h"
using namespace std;

Boton::Boton(int nroDeBoton, Mapa * mapa) : PanelContenedor("Boton", getRutaImgDefecto(nroDeBoton),
getRutaImgMouseDown(nroDeBoton), getRutaImgMouseOver(nroDeBoton)) {
    pf_rutinaMouseDown = NULL;
    pf_rutinaMouseOver = NULL;
    pf_rutinaMouseUp = NULL;
    this->mapa = mapa;
    presionado = false;
}

Boton::~Boton() {}

bool Boton::esta_presionado() const {
    return presionado;
}

void Boton::desapretar() {
    presionado = false;
}

void Boton::evento_MouseDown(Coordenada punto) {
    PanelContenedor::evento_MouseDown(punto);
    ventana->desclickearBotones(this->getId());
    if(presionado == false)
        presionado = true;
    else
        presionado = false;
    if(pf_rutinaMouseDown != NULL)
        pf_rutinaMouseDown(punto, this);
}

void Boton::evento_MouseUp(Coordenada punto) {
    PanelContenedor::evento_MouseUp(punto);

    if(pf_rutinaMouseUp != NULL)
        pf_rutinaMouseUp(punto, this);
}

void Boton::evento_MouseOver(Coordenada punto) {
    PanelContenedor::evento_MouseOver(punto);
    if(pf_rutinaMouseOver != NULL)
        pf_rutinaMouseOver(punto, this);
}

void Boton::setRutinaMouseDown(void (*rutina)(Coordenada, Boton *)) {
    pf_rutinaMouseDown = rutina;
}

void Boton::setRutinaMouseUp(void (*rutina)(Coordenada, Boton *)) {
    pf_rutinaMouseUp = rutina;
}

void Boton::setRutinaMouseOver(void (*rutina)(Coordenada, Boton *)) {
    pf_rutinaMouseOver = rutina;
}

Mapa * Boton::getMapa() const {
    return mapa;
}

void Boton::enviarSenialSalida() {
    ventana->salir = true;
}

void Boton::enviarSenialSms() {
```

```
if(!ventana->sms)
    ventana->sms = true;
else
    ventana->sms = false;
}

void Boton::enviarSenialSimulacion() {
    ventana->pideSimulacion = true;
}

bool Boton::getSmsState(){
    return ventana->sms;
}

long Boton::pedirTiempoResolucion() {
    return ventana->getTiempoResolucion();
}

string Boton::getRutaImgDefecto(int nroDeBoton) {
    switch(nroDeBoton) {
        case MASA: return "./ima/botones/icon_level_off.png";
        case RUEDA: return "./ima/botones/PlataformaMetalica_Defecto.bmp";
        case PUNTO_FIJO: return "./ima/botones/icon_node_off.png";
        case BARRA: return "./ima/botones/icon_metal_bar_off.png";
        case PLATAFORMA: return "./ima/botones/icon_metal_sheet_off.png";
        case SOGA: return "./ima/botones/icon_elastic_off.png";
        case CINTA: return "./ima/botones/icon_cloth_off.png";
        case COHETE: return "./ima/botones/icon_rocket_off.png";
        case BORRAR_TODO: return "./ima/botones/icon_clear_off.png";
        case BORRAR_ELEMENTO: return "./ima/botones/icon_rubber_off.png";
        case PLAY: return "./ima/botones/icon_start_off.png";
        case SALIR: return "./ima/botones/icon_salir_off.png";
        case SMS: return "./ima/botones/icon_sms_off.png";
        case ZONA_LLEGADA: return "./ima/botones/icon_llegada_off.png";
        case VOID: return "./letras/vacio.bmp";
    }
    return "";
}

string Boton::getRutaImgMouseDown(int nroDeBoton) {
    switch(nroDeBoton) {
        case MASA: return "./ima/botones/icon_level_on.png";
        case RUEDA: return "./ima/botones/PlataformaMetalica_MouseDown.bmp";
        case PUNTO_FIJO: return "./ima/botones/icon_node_on.png";
        case BARRA: return "./ima/botones/icon_metal_bar_on.png";
        case PLATAFORMA: return "./ima/botones/icon_metal_sheet_on.png";
        case SOGA: return "./ima/botones/icon_elastic_on.png";
        case CINTA: return "./ima/botones/icon_cloth_on.png";
        case COHETE: return "./ima/botones/icon_rocket_on.png";
        case BORRAR_TODO: return "./ima/botones/icon_clear_on.png";
        case BORRAR_ELEMENTO: return "./ima/botones/icon_rubber_on.png";
        case PLAY: return "./ima/botones/icon_start_on.png";
        case SALIR: return "./ima/botones/icon_salir_on.png";
        case SMS: return "./ima/botones/icon_sms_on.png";
        case ZONA_LLEGADA: return "./ima/botones/icon_llegada_on.png";
        case VOID: return "./letras/vacio.bmp";
    }
    return "";
}

string Boton::getRutaImgMouseOver(int nroDeBoton) {
    switch(nroDeBoton) {
        case MASA: return "./ima/botones/icon_level_over.png";
        case RUEDA: return "./ima/botones/PlataformaMetalica_MouseOver.bmp";
        case PUNTO_FIJO: return "./ima/botones/icon_node_over.png";
        case BARRA: return "./ima/botones/icon_metal_bar_over.png";
        case PLATAFORMA: return "./ima/botones/icon_metal_sheet_over.png";
    }
}
```

```
        case SOGA: return "./ima/botoness/icon_elastic_over.png";
        case CINTA: return "./ima/botoness/icon_cloth_over.png";
        case COHETE: return "./ima/botoness/icon_rocket_over.png";
        case BORRAR_TODO: return "./ima/botoness/icon_clear_over.png";
        case BORRAR_ELEMENTO: return "./ima/botoness/icon_rubber_over.png";
        case PLAY: return "./ima/botoness/icon_start_over.png";
        case SALIR: return "./ima/botoness/icon_salir_over.png";
        case SMS: return "./ima/botoness/icon_sms_over.png";
        case ZONA_LLEGADA: return "./ima/botoness/icon_llegada_over.png";
        case VOID: return "./letras/vacio.bmp";
    }

    return "";
}
```

```
#include "Cinta.h"

Cinta::Cinta() :
    ElementoLongitudinal("Longitudinal:Cinta", Coordenada(long(10), long(10)), 25, 0, 10) {}

Cinta::Cinta(Cordenada posicion, ulong magnitud, long angulo, long ancho) :
    ElementoLongitudinal("Longitudinal:Cinta", posicion, magnitud, angulo,
                           ancho) {}

Cinta::~Cinta() {}
```

```
#include "client_ChatDoble.h"
#include <iostream>
#include <fstream>
using namespace std;

#define INICIO_CHAT_X 0.965
#define INICIO_CHAT_Y 0.1
#define TAM_LETRA_X 0.025
#define TAM_LETRA_Y 0.05

#define INICIO_FONDO_X 0.9
#define INICIO_FONDO_Y 0.3
#define TAM_FONDO_X 0.75
#define TAM_FONDO_Y 1.2

#define INICIO_FONDO_RIVAL_X 0.57
#define INICIO_FONDO_RIVAL_Y 1.0
#define TAM_FONDO_RIVAL_X 1.1
#define TAM_FONDO_RIVAL_Y 0.7
#define ANCHO_LINEA 0.005

ChatDoble::ChatDoble()
{
    visible = true;
    salir = false;
    nuevoMovimiento = false;
    simular = false;
    bloquear = false;
}

bool ChatDoble::debeSimular()
{
    bool retorno;
    m.lock();
    retorno = simular;
    m.unlock();
    return retorno;
}

void ChatDoble::setSimular(bool valor)
{
    m.lock();
    simular = valor;
    m.unlock();
}

bool ChatDoble::debeBloquear()
{
    bool retorno;
    m.lock();
    retorno = bloquear;
    m.unlock();
    return retorno;
}

void ChatDoble::setBloquear(bool valor)
{
    m.lock();
    bloquear = valor;
    m.unlock();
}

void ChatDoble::cargarTexturas()
{
    string path;

    char i;
    // carga texturas minúsculas
    for (i = 'a' ; i <= 'z' ; i++)
    {
```

```
    path = "letras/letra__";
    path += i;
    path += ".bmp";
    loadTexture(path.c_str(),i);
}
// carga texturas mayúsculas
for (i = 'A' ; i <= 'Z' ; i++)
{
    path = "letras/letra__";
    path += i;
    path += ".bmp";
    loadTexture(path.c_str(),i);
}
// carga texturas números
for (i = '0' ; i <= '9' ; i++)
{
    path = "letras/letra__";
    path += i;
    path += ".bmp";
    loadTexture(path.c_str(),i);
}
// carga caracteres especiales
loadTexture("letras/vacio.bmp",' ');
loadTexture("letras/letra_dos_puntos.bmp",';');
loadTexture("letras/letra_punto.bmp",'.');
loadTexture("letras/letra_coma.bmp",' ,');
loadTexture("letras/letra_admiracion.bmp",'!');
loadTexture("letras/letra_interrogacion.bmp",'?');
loadTexture("letras/letra_admiracion_bajo.bmp", (char)161);
loadTexture("letras/letra_amp.bmp",'&');
loadTexture("letras/letra_arroba.bmp",'@');
loadTexture("letras/letra_asterisco.bmp",'*');
loadTexture("letras/letra_barra.bmp",'/');
loadTexture("letras/letra_comillas.bmp",'""');
loadTexture("letras/letra_guion.bmp",' -');
loadTexture("letras/letra_guion_bajo.bmp",' _');
loadTexture("letras/letra_interrogacion_bajo.bmp", (char)191);
loadTexture("letras/letra_not.bmp", (char)172);
loadTexture("letras/letra_numeral.bmp",'#');
loadTexture("letras/letra_parentesis_izq.bmp",'(');
loadTexture("letras/letra_parentesis_der.bmp",')');
loadTexture("letras/letra_pesos.bmp",'$');
loadTexture("letras/letra_porcentaje.bmp",'%');

//carga fondos y cursor
loadTexture("negro.bmp",' +');
loadTexture("letras/cursor.bmp",' =');
loadTexture("letras/fondo.png",' >');
}

bool ChatDoble::agregarLetra(char letra)
{
    if (buffer.length() == MAX_CARACTERES)
        return false; // no hay más lugar

    buffer += letra;
    return true;
}

bool ChatDoble::quitarLetra()
{
    if (buffer.length() == 0)
        return false;

    buffer = buffer.substr(0, buffer.length() - 1); //borra una letra
    return true;
}

void ChatDoble::agregarLinea(const std::string& linea)
{
```

```
if (lineas.size() == MAX_LINEAS)
    lineas.erase(lineas.begin());
lineas.push_back(linea);

}

void ChatDoble::imprimirChat()
{
    if (!visible)
        return;

    imprimirFondo();

    unsigned int i;
    string linea;
    float x = INICIO_CHAT_X;
    float y = INICIO_CHAT_Y;

    m.lock();

    // imprime líneas
    vector<string>::iterator itLineas = lineas.begin();
    while (itLineas != lineas.end())
    {
        linea = *itLineas;
        for ( i = 0 ; i < linea.length() ; i++)
        {
            imprimirCaracter(linea[i],x,y);
            x += TAM_LETRA_X;
        }
        x = INICIO_CHAT_X;
        y -= TAM_LETRA_Y;
        itLineas++;
    }
    // imprime buffer
    y = INICIO_CHAT_Y - (MAX_LINEAS+3) * TAM_LETRA_Y;
    for ( i = 0 ; i < buffer.length() ; i++)
    {
        imprimirCaracter(buffer[i],x,y);
        x += TAM_LETRA_X;
    }
    imprimirCaracter('=',x,y);

    m.unlock();
}

void ChatDoble::imprimirFondoRival()
{
    map<char,GLuint>::iterator it = texturas.find('+');
    if (it == texturas.end())
        return;

    GLuint textura = it->second;

    glBindTexture( GL_TEXTURE_2D, textura);

    //Línea izquierda
    glBegin(GL_QUADS);
    {
        glTexCoord2i( 0, 0 );
        glVertex3f( INICIO_FONDO_RIVAL_X, INICIO_FONDO_RIVAL_Y, 0.0f );
        glTexCoord2i( 0, 1 );
        glVertex3f( INICIO_FONDO_RIVAL_X, INICIO_FONDO_RIVAL_Y - TAM_FONDO_RIVAL_Y, 0.0f );
        glTexCoord2i( 1, 1 );
        glVertex3f( INICIO_FONDO_RIVAL_X + ANCHO_LINEA, INICIO_FONDO_RIVAL_Y -
TAM_FONDO_RIVAL_Y, 0.0f );
        glTexCoord2i( 1, 0 );
        glVertex3f( INICIO_FONDO_RIVAL_X + ANCHO_LINEA, INICIO_FONDO_RIVAL_Y, 0.0f );
    }
    glEnd();
}
```

```

//Linea derecha
glBegin(GL_QUADS);
{
    glTexCoord2i( 0, 0 );
    glVertex3f( INICIO_FONDO_RIVAL_X + TAM_FONDO_RIVAL_X, INICIO_FONDO_RIVAL_Y, 0.0f );
    glTexCoord2i( 0, 1 );
    glVertex3f( INICIO_FONDO_RIVAL_X + TAM_FONDO_RIVAL_X, INICIO_FONDO_RIVAL_Y -
TAM_FONDO_RIVAL_Y, 0.0f );
    glTexCoord2i( 1, 1 );
    glVertex3f( INICIO_FONDO_RIVAL_X + TAM_FONDO_RIVAL_X + ANCHO_LINEA,
INICIO_FONDO_RIVAL_Y - TAM_FONDO_RIVAL_Y, 0.0f );
    glTexCoord2i( 1, 0 );
    glVertex3f( INICIO_FONDO_RIVAL_X + TAM_FONDO_RIVAL_X + ANCHO_LINEA,
INICIO_FONDO_RIVAL_Y, 0.0f );
}
glEnd();

//Linea arriba
glBegin(GL_QUADS);
{
    glTexCoord2i( 0, 0 );
    glVertex3f( INICIO_FONDO_RIVAL_X, INICIO_FONDO_RIVAL_Y, 0.0f );
    glTexCoord2i( 0, 1 );
    glVertex3f( INICIO_FONDO_RIVAL_X, INICIO_FONDO_RIVAL_Y - ANCHO_LINEA, 0.0f );
    glTexCoord2i( 1, 1 );
    glVertex3f( INICIO_FONDO_RIVAL_X + TAM_FONDO_RIVAL_X + ANCHO_LINEA,
INICIO_FONDO_RIVAL_Y - ANCHO_LINEA, 0.0f );
    glTexCoord2i( 1, 0 );
    glVertex3f( INICIO_FONDO_RIVAL_X + TAM_FONDO_RIVAL_X + ANCHO_LINEA,
INICIO_FONDO_RIVAL_Y, 0.0f );
}
glEnd();

//Linea abajo
glBegin(GL_QUADS);
{
    glTexCoord2i( 0, 0 );
    glVertex3f( INICIO_FONDO_RIVAL_X, INICIO_FONDO_RIVAL_Y - TAM_FONDO_RIVAL_Y, 0.0f );
    glTexCoord2i( 0, 1 );
    glVertex3f( INICIO_FONDO_RIVAL_X, INICIO_FONDO_RIVAL_Y - TAM_FONDO_RIVAL_Y -
ANCHO_LINEA, 0.0f );
    glTexCoord2i( 1, 1 );
    glVertex3f( INICIO_FONDO_RIVAL_X + TAM_FONDO_RIVAL_X + ANCHO_LINEA,
INICIO_FONDO_RIVAL_Y - TAM_FONDO_RIVAL_Y - ANCHO_LINEA, 0.0f );
    glTexCoord2i( 1, 0 );
    glVertex3f( INICIO_FONDO_RIVAL_X + TAM_FONDO_RIVAL_X + ANCHO_LINEA,
INICIO_FONDO_RIVAL_Y - TAM_FONDO_RIVAL_Y, 0.0f );
}
glEnd();

}

void ChatDoble::imprimirFondo()
{
    map<char,GLuint>::iterator it = texturas.find('>');
    if (it == texturas.end())
        return;

    GLuint textura = it->second;

    glBindTexture( GL_TEXTURE_2D, textura);

    glBegin(GL_QUADS);
    {
        glTexCoord2i( 0, 0 );
        glVertex3f( INICIO_FONDO_X, INICIO_FONDO_Y, 0.0f );
        glTexCoord2i( 0, 1 );
        glVertex3f( INICIO_FONDO_X, INICIO_FONDO_Y - TAM_FONDO_Y, 0.0f );
    }
}

```

```
glTexCoord2i( 1, 1 );
glVertex3f( INICIO_FONDO_X + TAM_FONDO_X, INICIO_FONDO_Y - TAM_FONDO_Y, 0.0f );
glTexCoord2i( 1, 0 );
glVertex3f( INICIO_FONDO_X + TAM_FONDO_X, INICIO_FONDO_Y, 0.0f );
}
glEnd();

imprimirFondoRival();

}

void ChatDoble::imprimirCaracter(char letra, float x, float y)
{

map<char,GLuint>::iterator it = texturas.find(letra);
if (it == texturas.end())
    return;

GLuint textura = it->second;

glBindTexture( GL_TEXTURE_2D, textura);

glBegin(GL_QUADS);
{
    glTexCoord2i( 0, 0 );
    glVertex3f( x, y, 0.0f );
    glTexCoord2i( 0, 1 );
    glVertex3f( x, y - TAM_LETRA_Y, 0.0f );
    glTexCoord2i( 1, 1 );
    glVertex3f( x + TAM_LETRA_X, y - TAM_LETRA_Y, 0.0f );
    glTexCoord2i( 1, 0 );
    glVertex3f( x + TAM_LETRA_X, y, 0.0f );
}
glEnd();
}

bool ChatDoble::enter()
{
    if (buffer.length() == 0)
        return false;

    m.lock();
    entradas.push_back("C"+buffer);
    m.unlock();

    buffer.clear();

    return true;
}

void ChatDoble::agregarMensaje(const std::string& mensaje)
{
    m.lock();
    entradas.push_back(mensaje);
    m.unlock();
}

bool ChatDoble::obtenerEntrada(std::string& entrada)
{
    bool retorno = true;
    m.lock();
    if (entradas.empty())
        retorno = false;
    else
    {
        entrada = entradas[0];
        entradas.erase(entradas.begin());
    }
    m.unlock();
    return retorno;
}
```

```
char ChatDoble::esLetra(SDL_Event evento)
{
    if (((evento.key.keysym.unicode >= 'a') && (evento.key.keysym.unicode <= 'z'))
        ||
        ((evento.key.keysym.unicode >= 'A') && (evento.key.keysym.unicode <= 'Z'))
        ||
        ((evento.key.keysym.unicode >= '0') && (evento.key.keysym.unicode <= '9'))
        ||
        (evento.key.keysym.unicode == '\"') || (evento.key.keysym.unicode == '#')
        ||
        (evento.key.keysym.unicode == '%') || (evento.key.keysym.unicode == '$')
        ||
        (evento.key.keysym.unicode == '/') || (evento.key.keysym.unicode == 172)
        ||
        (evento.key.keysym.unicode == '.') || (evento.key.keysym.unicode == ':')
        ||
        (evento.key.keysym.unicode == '-') || (evento.key.keysym.unicode == ',')
        ||
        (evento.key.keysym.unicode == '_') || (evento.key.keysym.unicode == '(')
        ||
        (evento.key.keysym.unicode == '*') || (evento.key.keysym.unicode == ')')
        ||
        (evento.key.keysym.unicode == '@') || (evento.key.keysym.unicode == '&')
        ||
        (evento.key.keysym.unicode == 161) || (evento.key.keysym.unicode == 191) ||
        (evento.key.keysym.unicode == ' ')
        ||
        (evento.key.keysym.unicode == '!') || (evento.key.keysym.unicode == '?'))
    {
        return evento.key.keysym.unicode;
    }
    else
        return -1;
}

void ChatDoble::loadTexture(const char * path, char letra) {
    GLuint texture;
    GLint nOfColors;
    GLenum texture_format = 0;
    SDL_Surface *surface;

    if ( (surface = IMG_Load(path)) ) {

        if ( (surface->w & (surface->w - 1)) != 0 ) {
            printf("ERROR en imagen, el ancho no es potencia de 2\n");
        }

        if ( (surface->h & (surface->h - 1)) != 0 ) {
            printf("ERROR en imagen, el alto no es potencia de 2\n");
        }

        nOfColors = surface->format->BytesPerPixel;
        if (nOfColors == 4)
        {
            if (surface->format->Rmask == 0x000000ff)
                texture_format = GL_RGBA;
            else
                texture_format = GL_BGRA;
        } else if (nOfColors == 3)
        {
            if (surface->format->Rmask == 0x000000ff)
                texture_format = GL_RGB;
            else
                texture_format = GL_BGR;
        } else {
            printf("ERROR truecolor\n");
        }
    }

    glGenTextures( 1, &texture );
    glBindTexture( GL_TEXTURE_2D, texture );

    glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR );
    glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR );

    glTexImage2D( GL_TEXTURE_2D, 0, nOfColors, surface->w, surface->h, 0,
                  texture_format, GL_UNSIGNED_BYTE, surface->pixels );
    texturas.insert(make_pair(letra, texture));
}
```

```
}

else {
    printf("SDL no pudo abrir imagen: %s\n", SDL_GetError());
    SDL_Quit();
}

void ChatDoble::cambiarVisible()
{
    visible = !visible;
}

bool ChatDoble::debeSalir()
{
    bool retorno;
    m.lock();
    retorno = salir;
    m.unlock();
    return retorno;
}

void ChatDoble::setSalir(bool valor)
{
    m.lock();
    salir = valor;
    m.unlock();
}

void ChatDoble::cambiarMovimiento(const std::string& xml)
{
    m.lock();
    nuevoMovimiento = true;
    movimiento = xml;
    m.unlock();
}

bool ChatDoble::obtenerMovimiento(std::string& xml)
{
    bool retorno = true;
    m.lock();
    if (nuevoMovimiento)
    {
        xml = movimiento;
        nuevoMovimiento = false;
    }
    else
        retorno = false;
    m.unlock();
}

return retorno;
}
```

```
#include "client_ContactaServer.h"
using namespace std;

/* Mensajes */
#define MENSAJE_CAE_SERVER "Se cayó el server"
#define MENSAJE_SE_DESCONECTO "Se ha desconectado"
#define MENSAJE_DESCONECCION "/desconectar"
#define ERROR_USUARIO_EXISTENTE "Nombre de usuario existente. \n Conéctese nuevamente con otro nombre de usuario."

ContactaServer::ContactaServer()
{
    conectado = false;
    socket.create(); // crea socket
}

ContactaServer::~ContactaServer()
{
    if (socketConectado())
        desconectarSocket();
}

void ContactaServer::run()
{
    string mensaje;
    while (estaVivo())
    {
        if (socketConectado())
        {
            if (!socket.recv(mensaje))
            {
                //Se desconectó del Server
                mensaje = "DSe desconectó el server";
                setConectado(false);
                socket.close();
            }
            if (mensaje.compare("SUsuario existente") == 0)
            {
                //Ya existe el nombre de usuario del cliente en el servidor
                mensaje = "D"+mensaje;
                socket.close();
                setConectado(false);
            }

            if (mensaje.compare("D") == 0)
            {
                //Debe desconectarse
                socket.close();
                setConectado(false);
            }

            //Carga el mensaje en la lista de Mensajes Recibidos
            bloquearEstado();
            mensajesRecibidos.push_back(mensaje);
            desbloquearEstado();
        }
    }
}

bool ContactaServer::conectar(const string& host, const int puerto)
{
    if (socketConectado())
        desconectarSocket();
    if (socket.connect(host,puerto))
    {
        setConectado(true);
```

```
        return true;
    }

    return false;
}

bool ContactaServer::socketConectado()
{
    bool retorno;
    bloquearEstado();
    retorno = conectado;
    desbloquearEstado();
    return retorno;
}

void ContactaServer::desconectarSocket()
{
    if (!socketConectado())
        return;

    //Envía un mensaje de desconexión al server
    enviarMensaje("D");
    sleep(0);

    //Espera que vuelva el mensaje de desconexión desde el server y se destrabe el recv
    while(socketConectado())
        sleep(0);
}

void ContactaServer::setConectado(bool valor)
{
    bloquearEstado();
    conectado = valor;
    desbloquearEstado();
}

bool ContactaServer::obtenerMensajeRecibido(string& mensaje)
{
    bool retorno;
    bloquearEstado();
    if (!mensajesRecibidos.empty())
    {
        retorno = true;

        //Toma el primero de la lista y lo saca de la lista
        mensaje = mensajesRecibidos[0];
        mensajesRecibidos.erase(mensajesRecibidos.begin());
    }
    else
        retorno = false;
    desbloquearEstado();
    return retorno;
}

bool ContactaServer::enviarMensaje(const string& mensaje)
{
    if (socketConectado())
        return socket.send(mensaje);
    return false;
}

void ContactaServer::agregarMensaje(const string& mensaje)
{
    bloquearEstado();
    mensajesRecibidos.push_back(mensaje);
```

```
    desbloquearEstado();  
}  
  
void ContactaServer::vaciarMensajesRecibidos()  
{  
    bloquearEstado();  
    mensajesRecibidos.clear();  
    desbloquearEstado();  
}
```

```
#include "client_HiloMovimientos.h"
#include "ParserXML.h"
using namespace std;

#define TIEMPO_RETARDO 1000000 //en microsegundos
#define CANT_INTERVALOS 10

HiloMovimientos::HiloMovimientos(Mapa* mapa)
{
    listo = false;
    this->mapa = mapa;
}

void HiloMovimientos::run()
{
    ParserXML par;
    string xmlEspera;
    unsigned int intervalos = 0;
    while (estaVivo())
    {
        if (intervalos == CANT_INTERVALOS)
        {
            bloquearEstado();
            xmllist = xmlEspera; //ya revela el movimiento
            listo = true;
            desbloquearEstado();
            intervalos = 0;
            par.obtenerXMLMemoria(xmlEspera, mapa); //obtiene un nuevo movimiento a revelar
en el futuro
        }
        else
        {
            intervalos++;
            usleep(TIEMPO_RETARDO / CANT_INTERVALOS); //retardo
        }
    }
}

bool HiloMovimientos::obtenerMovimiento(std::string& xml)
{
    bool retorno = true;
    bloquearEstado();
    if (!listo)
        retorno = false;
    else
    {
        xml = this->xmllist;
        listo = false; //empieza de nuevo
    }
    desbloquearEstado();
    return retorno;
}
```

```
#include "client_InterfazBatalla.h"
#include "ParserXML.h"
#include "InterfazEdicion.h"
#include <fstream>
#include <glib.h>
#include <glib/gprintf.h>
#include <gtk/gtk.h>
using namespace std;

InterfazBatalla::InterfazBatalla(ContactaServer* contacto)
{
    this->contacto = contacto;
    jugando = true;
}

void InterfazBatalla::iniciarBatalla(ChatDoble* chat)
{
    this->chat = chat;

    //Recibo los archivos
    bool recibidojpg = false;
    bool recibidoxml = false;
    bool empezar = false;
    string mensaje;
    string archivo;

    ParserXML par;

    while ((!recibidojpg) || (!recibidoxml))
    {
        if (contacto->obtenerMensajeRecibido(mensaje))
        {

            //Pregunta por tipo lista
            if (mensaje[0] == 'A')
            {

                if (!recibidojpg)
                {
                    desconvertir(archivo,mensaje.substr(1,mensaje.length()-1));
                    ofstream arch("escenario.jpg");
                    arch << archivo;
                    arch.close();
                    recibidojpg = true;
                }
                else
                {
                    desconvertir(archivo,mensaje.substr(1,mensaje.length()-1));
                    mapa = par.obtenerMapaMemoria(archivo);
                    recibidoxml = true;
                }
            }
        }
        sleep(0);
    }

    //Envia mensaje de lista
    contacto->enviarMensaje("L");

    //Espera mensaje de jugar
    while (!empezar)
    {
        if (contacto->obtenerMensajeRecibido(mensaje))
        {
            //Pregunta por tipo lista
            if (mensaje[0] == 'J')
                empezar = true;
        }
        sleep(0);
    }
}
```

```
movimientos = new HiloMovimientos(mapa);
movimientos->start();

}

char InterfazBatalla::obtenerValor (char hexa)
{
    switch (hexa)
    {
        case '0': return 0;
        case '1': return 1;
        case '2': return 2;
        case '3': return 3;
        case '4': return 4;
        case '5': return 5;
        case '6': return 6;
        case '7': return 7;
        case '8': return 8;
        case '9': return 9;
        case 'a': return 10;
        case 'b': return 11;
        case 'c': return 12;
        case 'd': return 13;
        case 'e': return 14;
        case 'f': return 15;
    }
    return -1;
}

void InterfazBatalla::run()
{
    InterfazEdicion interfaz(mapa);
    interfaz.comenzar(chat);

    //set puntos;
    puntos = mapa->getPuntos();

    if (puntos < 0)
        // huyo
        contacto->enviarMensaje("H");

    movimientos->terminar();
    movimientos->join();

    setJugando(false);

}

void InterfazBatalla::desconvertir(string& resultado, const string& original)
{
    unsigned int i = 0;
    char buf;
    char bufaux;
    resultado = "";
    for ( ; i < original.length() ; i += 2 )
    {
        buf = obtenerValor(original[i]);
        buf = buf << 4;
        bufaux = obtenerValor(original[i+1]);
        buf = buf | bufaux;
        resultado += buf;
    }
}

bool InterfazBatalla::analizarMensajesRecibidos()
{
    string mensaje;
    string xml;
```

```
GtkWidget *alerta;
GtkWidget* label;
GtkWidget* boton;

while (chat->obtenerEntrada(mensaje))
{
    contacto->enviarMensaje(mensaje);
}

// analizar los movimientos
if (movimientos->obtenerMovimiento(xml))
    contacto->enviarMensaje("M" + xml);

while (contacto->obtenerMensajeRecibido(mensaje))
{

    switch (mensaje[0])
    {
        case 'C': //chat
            chat->agregarLinea(mensaje.substr(1,mensaje.length() - 1));
            break;
        case 'D': //desconexión
            chat->setSalir(true);
            return false; //Se desconectó el servidor
        case 'S':
            //simular
            chat->setSimular(true);
            break;
        case 'M': //movimiento
            chat->cambiarMovimiento(mensaje.substr(1,mensaje.length() - 1));
            break;
        case 'H': //huyó
            //avisa a la ventana que debe morir
            chat->setSalir(true);

            mapa->setPuntos(100); // premio consuelo porque escapa el otro

            this->join();

            alerta = gtk_dialog_new ();
            label = gtk_label_new ("Su rival abandonó la partida");
            gtk_window_set_title (GTK_WINDOW (alerta), "TatuCarreta");

            gtk_box_pack_start (GTK_BOX (GTK_DIALOG (alerta)->vbox), label, FALSE,
FALSE, 0);

            boton = gtk_button_new_from_stock("Aceptar");
            gtk_box_pack_start (GTK_BOX (GTK_DIALOG (alerta)->action_area), boton,
FALSE, FALSE, 0);

            gtk_window_set_position(GTK_WINDOW(alerta),GTK_WIN_POS_CENTER);

            gtk_signal_connect_object (GTK_OBJECT (boton), "clicked",
GTK_SIGNAL_FUNC (gtk_widget_destroy), GTK_OBJECT
                (alerta));

            gtk_widget_show_all (alerta);

            break;

        case 'I':
            //alguien gano
            chat->setSalir(true);

            this->join();

            alerta = gtk_dialog_new ();
            label = gtk_label_new ((mensaje.substr(1,mensaje.length()-1)).c_str());
            gtk_window_set_title (GTK_WINDOW (alerta), "TatuCarreta");
            gtk_box_pack_start (GTK_BOX (GTK_DIALOG (alerta)->vbox), label, FALSE,
FALSE, 0);
    }
}
```

```
boton = gtk_button_new_from_stock("Aceptar");
gtk_box_pack_start(GTK_BOX(GTK_DIALOG(alerta)->action_area), boton,
FALSE, FALSE, 0);

    gtk_window_set_position(GTK_WINDOW(alerta), GTK_WIN_POS_CENTER);

    gtk_signal_connect_object (GTK_OBJECT (boton), "clicked",
GTK_SIGNAL_FUNC (gtk_widget_destroy), GTK_OBJECT
(alerta));

    gtk_widget_show_all (alerta);

    break;
case 'X': //bloquear
chat->setBloquear(true);
break;
}
}

return true;
}

bool InterfazBatalla::estaJugando()
{
    bool ret;
    bloquearEstado();
    ret = jugando;
    desbloquearEstado();
    return ret;
}

int InterfazBatalla::getPuntos()
{
    int ret;
    bloquearEstado();
    ret = puntos;
    desbloquearEstado();
    return ret;
}

void InterfazBatalla::setJugando(bool valor)
{
    bloquearEstado();
    jugando = valor;
    desbloquearEstado();
}
```

```
#include "client_InterfazGrafica.h"
#include <iostream>
#include <string>
#include <vector>
#include <fstream>
#include "ParserXML.h"
#include "InterfazEdicion.h"
#include "Mapa.h"
#include <fstream>
using namespace std;

#define TITULO_VENTANA "TatuCarreta Run"
#define TEXTO_INICIAL "Bienvenido! \n"

/* Estructuras de datos auxiliares para la invocación de funciones CallBack
 * con parámetros determinados. */

typedef struct datosConexion
{
    GtkWidget* entradaIp;
    GtkWidget* entradaPuerto;
    GtkWidget* entradaNombre;
    GtkWidget* eleccion;
    GtkWidget* ventanaConexion;
    InterfazGrafica* pthis;
} datosConexion;

typedef struct datosBatalla
{
    GtkWidget* ventanaEleccion;
    InterfazGrafica* pthis;
} datosBatalla;

typedef struct datosGravedad
{
    GtkWidget* ventanaGravedad;
    GtkWidget* escala;
} datosGravedad;

typedef struct datosDesafio
{
    GtkWidget* ventanaEleccion;
    InterfazGrafica* pthis;
    string rival;
} datosDesafio;

/* Constructor */
InterfazGrafica::InterfazGrafica()
{
    // inicializa atributos
    enBatalla = false;
    batalla = NULL;
    juegoSolo = NULL;
}

bool InterfazGrafica::textoVacio(const string& texto)
{
    unsigned int i;
    bool retorno = true;
    for (i = 0 ; i < texto.length() ; i++)
    {
        if (texto[i] != ' ')
        {
            retorno = false;
            break;
        }
    }
    return retorno;
}

void InterfazGrafica::obtenerTokens(std::string cadena, std::vector<std::string>& tokens)
```

```
{  
    string::size_type inicio = 0; //inicio del token  
    string::size_type fin; //fin del token  
    string token;  
  
    while (inicio < cadena.length())  
    {  
        fin = cadena.find("\n", inicio);  
        if (fin == string::npos)  
            fin = cadena.length();  
        token = cadena.substr(inicio, fin - inicio);  
        tokens.push_back(token);  
        inicio = fin + 1;  
    }  
}  
  
void InterfazGrafica::rechazarBatalla(GtkWidget *widget, gpointer data)  
{  
    datosDesafio* datos = (datosDesafio*) data;  
    datos->pthis->contacto.enviarMensaje("R" + datos->rival); //envía rechazo  
    gtk_widget_destroy(datos->ventanaEleccion);  
}  
  
void InterfazGrafica::aceptarBatalla(GtkWidget *widget, gpointer data)  
{  
    datosDesafio* datos = (datosDesafio*) data;  
    datos->pthis->contacto.enviarMensaje("A" + datos->rival); //envía aceptación  
    gtk_widget_destroy(datos->ventanaEleccion);  
}  
  
void InterfazGrafica::iniciarBatalla(InterfazGrafica* pthis)  
{  
    pthis->enBatalla = true;  
    pthis->batalla = new InterfazBatalla(&(pthis->contacto)); //genera InterfazBatalla  
    pthis->chat = new ChatDoble(); //genera ChatDoble  
    pthis->batalla->iniciarBatalla(pthis->chat); //inicializa Batalla  
    pthis->batalla->start(); //corre batalla  
}  
  
void InterfazGrafica::apretar_boton_conectar_auto(GtkWidget *widget, gpointer data)  
{  
    InterfazGrafica* pthis = (InterfazGrafica*) data;  
    string ip = "127.0.0.1";  
    string puerto = "3000";  
    string nombre = "Axel";  
  
    string leyendaAlerta;  
    if (pthis->contacto.conectar(ip, atoi(puerto.c_str())))  
    {  
        pthis->contacto.vaciarMensajesRecibidos();  
        pthis->contacto.enviarMensaje("EAxel");  
  
        string respuesta;  
        while (true)  
        {  
            if (pthis->contacto.obtenerMensajeRecibido(respuesta))  
                break;  
            else  
                sleep(0);  
        }  
        if (respuesta.compare("SSe ha conectado exitosamente") == 0)  
        {  
            string leyendaEtiqueta = "Conectado a servidor " + ip + " con nombre  
de usuario " + nombre;  
        }  
    }  
}
```

```
        gtk_label_set_text(GTK_LABEL(pthis->etiqueta),leyendaEtiqueta.c_str());
    }
    mostrarAlerta(respuesta.substr(1, respuesta.length()-1),pthis->ventana);
    if (respuesta.compare("DSUsuario existente") == 0)
        mostrarAlerta("Usuario Existente",pthis->ventana);

}
else
    mostrarAlerta("No se pudo conectar al servidor",pthis->ventana);
}

void InterfazGrafica::mostrarAlerta(string leyenda, GtkWidget* ventanaPadre)
{
    GtkWidget *alerta = gtk_dialog_new ();
    GtkWidget* label = gtk_label_new (leyenda.c_str());

    gtk_window_set_title (GTK_WINDOW (alerta), "TatuCarreta");

    gtk_box_pack_start (GTK_BOX (GTK_DIALOG (alerta)->vbox), label, FALSE, FALSE, 0);

    GtkWidget* boton = gtk_button_new_from_stock("Aceptar");
    gtk_box_pack_start (GTK_BOX (GTK_DIALOG (alerta)->action_area), boton, FALSE, FALSE, 0);

    gtk_window_set_transient_for (GTK_WINDOW(alerta),GTK_WINDOW(ventanaPadre));
    gtk_window_set_position(GTK_WINDOW(alerta),GTK_WIN_POS_CENTER_ON_PARENT);

    g_signal_connect(G_OBJECT(boton), "clicked", G_CALLBACK(cerrar_ventana), alerta);
    gtk_widget_show_all (alerta);
}

void InterfazGrafica::elegirAdversario(GtkWidget *clist, gint row, gint column, GdkEventButton *event,
gpointer data)
{
    //Cast de los datos
    datosBatalla* datos = (datosBatalla*) data;

    //Obtengo el valor elegido
    gchar *elegido;
    gtk_clist_get_text(GTK_CLIST(clist), row, column, &elegido);

    string mensaje = " Ha solicitado juego con ";
    mensaje += elegido;

    datos->pthis->agregarMensajeAMostrar(mensaje);

    //Envio solicitud de juego al servidor
    mensaje = "B";
    mensaje += elegido;
    datos->pthis->contacto.enviarMensaje(mensaje.c_str());

    gtk_widget_destroy(datos->ventanaEleccion);
}

void InterfazGrafica::jugar_solo(GtkWidget *widget, gpointer data)
{
    InterfazGrafica* p = (InterfazGrafica*) data;

    if (!p->contacto.socketConectado())
        return ;

    Mapa* mapa = NULL;
    ParserXML par;

    p->contacto.enviarMensaje("J");

    //Recibo los archivos
    bool recibidojpg = false;
```

```
bool recibidoxml = false;

string mensaje;
string archivo;

while ((!recibidojpg) || (!recibidoxml))
{
    if (p->contacto.obtenerMensajeRecibido(mensaje))
    {
        if (mensaje[0] == 'T')
        {
            mostrarAlerta("Ha conocido todos \n los niveles disponibles",p->ventana);
            break; //Termino todos los niveles
        }
        if (mensaje[0] == '0')
        {
            mostrarAlerta("El Servidor está manejando su máximo \n de partidas simultáneas. Intente un rato más tarde",p->ventana);
            break;
        }

        if (mensaje[0] == 'A')
        {
            if (!recibidojpg)
            {
                //guardo imagen de fondo
                InterfazBatalla::desconvertir(archivo,mensaje.substr(1,mensaje.length()-1));
                ofstream arch("escenario.jpg");
                arch << archivo;
                arch.close();
                recibidojpg = true;
            }
            else
            {
                InterfazBatalla::desconvertir(archivo,mensaje.substr(1,mensaje.length()-1));

                //Hacer el mapa en base al xml.
                mapa = par.obtenerMapaMemoria(archivo);
                recibidoxml = true;
            }
        }
    }
    sleep(0);
}

if ((recibidojpg) && (recibidoxml))
{
    p->juegoSolo = new InterfazJugarSolo(mapa);
    p->juegoSolo->start(); //inicia juego solitario
}

}

void InterfazGrafica::jugar_batalla(GtkWidget *widget, gpointer data)
{
    //Pide los usuarios conectados
    InterfazGrafica* p = (InterfazGrafica*) data;

    if (!p->contacto.socketConectado())
        return;
    p->adversariosActualizados = false;
    p->contacto.enviarMensaje("U");
    while (!p->adversariosActualizados)
        analizarMensajesRecibidos(p);

    //Busco los usuarios
```

```
vector<string> usuarios;
obtenerTokens(p->adversarios,usuarios);

//Analizo si hay usuarios conectados para desafiar
if (usuarios.empty())
    mostrarAlerta("No hay otros usuarios conectados",p->ventana);
else
{
    //Arma la ventana
    GtkWidget* vent = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title (GTK_WINDOW (vent), "Jugar Con...");
    GtkWidget* label = gtk_label_new ("Seleccione un rival entre los usuarios conectados");
    GtkWidget* contenedorVertical = gtk_vbox_new(FALSE, 10);
    GtkWidget* lista = gtk_clist_new ( 1 );

    gtk_window_set_default_size (GTK_WINDOW (vent), 300, 300);
    gtk_clist_set_selection_mode( GTK_CLIST(lista), GTK_SELECTION_SINGLE );
    gtk_box_pack_start (GTK_BOX (contenedorVertical), label, FALSE, FALSE, 0);
    gtk_box_pack_start (GTK_BOX (contenedorVertical), lista, TRUE, TRUE, 0);
    gtk_container_add(GTK_CONTAINER(vent), contenedorVertical);

    gtk_window_set_transient_for (GTK_WINDOW(vent),GTK_WINDOW(p->ventana));
    gtk_window_set_position(GTK_WINDOW(vent),GTK_WIN_POS_CENTER_ON_PARENT);

    //Agrego los usuarios a la lista
    unsigned int i;
    gchar* nomUsuario;
    for (i = 0 ; i < usuarios.size() ; i++)
    {
        nomUsuario = (gchar*) usuarios[i].c_str();
        gtk_clist_append( GTK_CLIST(lista), &nomUsuario );
    }

    //Datos para la siguiente pantalla
    datosBatalla* datos = new datosBatalla();
    datos->ventanaEleccion = vent;
    datos->pthis = p;

    //Conecto la señal de selección
    gtk_signal_connect(GTK_OBJECT(lista), "select_row", GTK_SIGNAL_FUNC(elegirAdversario),
datos);
    gtk_widget_show_all(vent);
}
}

void InterfazGrafica::apretar_boton_aceptar(GtkWidget *widget, gpointer data)
{
    InterfazGrafica* pthis = (InterfazGrafica*) data;
    string mensaje = gtk_entry_get_text(GTK_ENTRY(pthis->entrada));

    //Si no dice nada en la caja de texto, no hace nada
    if (textoVacio(mensaje))
    {
        gtk_entry_set_text(GTK_ENTRY(pthis->entrada), "");
        return;
    }

    //Envía el mensaje
    if (!pthis->contacto.enviarMensaje("C" + mensaje))
        pthis->agregarMensajeAMostrar("No se pudo enviar el mensaje: " + mensaje);

    //Vacío la entrada
    gtk_entry_set_text(GTK_ENTRY(pthis->entrada), "");

}

void InterfazGrafica::cerrar_ventana(GtkWidget *widget, gpointer data)
{
```

```
GtkWidget* vent = (GtkWidget*) data;
gtk_widget_destroy(vent);
}

void InterfazGrafica::desconectar(GtkWidget *widget, gpointer data)
{
    InterfazGrafica* pthis = (InterfazGrafica*) data;
    if (pthis->contacto.socketConectado())
    {
        pthis->contacto.desconectarSocket();
        gtk_label_set_text(GTK_LABEL(pthis->etiqueta), "No está conectado a ningún servidor");
    }
}

void InterfazGrafica::conectar(GtkWidget *widget, gpointer data)
{
    datosConexion* datos = (datosConexion*) data;
    string ip = gtk_entry_get_text(GTK_ENTRY(datos->entradaIp));
    string puerto = gtk_entry_get_text(GTK_ENTRY(datos->entradaPuerto));
    string nombre = gtk_entry_get_text(GTK_ENTRY(datos->entradaNombre));

    //analizo si se completaron todos los datos
    if ((textoVacio(ip)) || (textoVacio(puerto)) || (textoVacio(nombre)))
        mostrarAlerta("Faltan ingresar datos", datos->ventanaConexion);
    else
    {
        string leyendaAlerta;
        if (datos->pthis->contacto.conectar(ip, atoi(puerto.c_str())))
        {
            datos->pthis->contacto.vaciarMensajesRecibidos();

            //analiza botón de elección
            if (GTK_TOGGLE_BUTTON (datos->eleccion)->active)
                datos->pthis->contacto.enviarMensaje("N"+nombre);
            else
                datos->pthis->contacto.enviarMensaje("E"+nombre);

            string respuesta;
            while (true)
            {
                if (datos->pthis->contacto.obtenerMensajeRecibido(respuesta))
                    break;
                else
                    sleep(0);
            }

            if (respuesta.compare("SSe ha conectado exitosamente") == 0)
            {
                gtk_widget_destroy(datos->ventanaConexion);
                string leyendaEtiqueta = "Conectado a servidor " + ip + " con nombre de usuario " + nombre;
                gtk_label_set_text(GTK_LABEL(datos->pthis->etiqueta), leyendaEtiqueta.c_str());
            }
            mostrarAlerta(respuesta.substr(1, respuesta.length()-1), datos->pthis->ventana);
            if (respuesta.compare("DSUsuario existente") == 0)
                mostrarAlerta("Usuario Existente", datos->ventanaConexion);
        }
        else
            mostrarAlerta("No se pudo conectar al servidor", datos->ventanaConexion);
    }
}
```

```
void InterfazGrafica::apretar_boton_conectar(GtkWidget *widget, gpointer data)
{
    InterfazGrafica* pthis = (InterfazGrafica*) data;

    // elementos de la ventana
    GtkWidget* ventanaConexion;
    GtkWidget* contenedorHorizontalIp = gtk_hbox_new(FALSE, 10);
    GtkWidget* contenedorHorizontalPuerto = gtk_hbox_new(FALSE, 10);
    GtkWidget* contenedorHorizontalNombre = gtk_hbox_new(FALSE, 10);
    GtkWidget* contenedorHorizontalEleccion = gtk_hbox_new(FALSE, 10);
    GtkWidget* contenedorVertical = gtk_vbox_new(FALSE, 10);
    GtkWidget* etiquetaIp = gtk_label_new("IP:");
    GtkWidget* etiquetaPuerto = gtk_label_new("Puerto:");
    GtkWidget* etiquetaNombre = gtk_label_new("Nombre:");
    GtkWidget* entradaIp = gtk_entry_new();
    gtk_entry_set_width_chars(GTK_ENTRY(entradaIp), 30);
    GtkWidget* entradaNombre = gtk_entry_new();
    gtk_entry_set_width_chars(GTK_ENTRY(entradaNombre), 30);
    GtkWidget* entradaPuerto = gtk_entry_new();
    gtk_entry_set_width_chars(GTK_ENTRY(entradaPuerto), 30);

    // botón de elección
    GtkWidget* eleccion = gtk_check_button_new_with_label ("Usuario Nuevo");
    gtk_box_pack_start (GTK_BOX(contenedorHorizontalEleccion),eleccion, FALSE, FALSE, 3);

    gtk_box_pack_start (GTK_BOX(contenedorHorizontalIp), etiquetaIp, FALSE, FALSE, 3);
    gtk_box_pack_start (GTK_BOX(contenedorHorizontalPuerto), etiquetaPuerto, FALSE, FALSE, 3);
    gtk_box_pack_start (GTK_BOX(contenedorHorizontalNombre), etiquetaNombre, FALSE, FALSE, 3);

    gtk_box_pack_start (GTK_BOX(contenedorHorizontalIp), entradaIp, TRUE, TRUE, 3);
    gtk_box_pack_start (GTK_BOX(contenedorHorizontalPuerto), entradaPuerto, TRUE, TRUE, 3);
    gtk_box_pack_start (GTK_BOX(contenedorHorizontalNombre), entradaNombre, TRUE, TRUE, 3);

    GtkWidget* boton = gtk_button_new_from_stock("gtk-ok");
    gtk_box_pack_start (GTK_BOX(contenedorHorizontalEleccion), boton, FALSE, FALSE, 3);

    ventanaConexion = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_window_set_transient_for (GTK_WINDOW(ventanaConexion),GTK_WINDOW(pthis->ventana));
    gtk_window_set_position(GTK_WINDOW(ventanaConexion),GTK_WIN_POS_CENTER_ON_PARENT);

    gtk_container_set_border_width(GTK_CONTAINER(ventanaConexion), 10); //Configura borde
    gtk_window_set_title (GTK_WINDOW (ventanaConexion), "Conexión a servidor"); //Pone título

    gtk_container_add(GTK_CONTAINER(ventanaConexion), contenedorVertical);
    gtk_box_pack_start (GTK_BOX(contenedorVertical), contenedorHorizontalIp, FALSE, TRUE, 1);
    gtk_box_pack_start (GTK_BOX(contenedorVertical), contenedorHorizontalPuerto, FALSE, TRUE, 1);
    gtk_box_pack_start (GTK_BOX(contenedorVertical), contenedorHorizontalNombre, FALSE, TRUE, 1);
    gtk_box_pack_start (GTK_BOX(contenedorVertical), contenedorHorizontalEleccion, FALSE, TRUE, 1);

    datosConexion * datos = new datosConexion();
    datos->entradaIp = entradaIp;
    datos->entradaPuerto = entradaPuerto;
    datos->entradaNombre = entradaNombre;
    datos->eleccion = eleccion;
    datos->ventanaConexion = ventanaConexion;
    datos->pthis = pthis;

    g_signal_connect(G_OBJECT(boton), "clicked", G_CALLBACK(conectar), datos);
    gtk_widget_show_all(ventanaConexion);
}

gboolean InterfazGrafica::on_delete_event(GtkWidget *widget, GdkEvent *event, gpointer data)
```

```
InterfazGrafica* pthis = (InterfazGrafica*) data;
pthis->contacto.desconectarSocket();
pthis->contacto.terminar();
pthis->contacto.join();

return FALSE;
}

void InterfazGrafica::destruir(GtkWidget *widget, gpointer data)
{
    gtk_main_quit();
}

void InterfazGrafica::propuestaBatalla(InterfazGrafica* pthis, const std::string& rival)
{
    string cadEtDesafio = rival;
    cadEtDesafio += " desea jugar una partida con usted.";

    // ventana de consulta
    GtkWidget* ventanaConsulta = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    GtkWidget* etiquetaDesafio = gtk_label_new(cadEtDesafio.c_str());
    GtkWidget* etiquetaPregunta = gtk_label_new("¿Acepta el desafío?");
    GtkWidget* botonSi = gtk_button_new_from_stock("Sí");
    GtkWidget* botonNo = gtk_button_new_from_stock("No");
    GtkWidget* contenedorHorizontal = gtk_hbox_new(FALSE, 10);
    GtkWidget* contenedorVertical = gtk_vbox_new(FALSE, 10);

    gtk_window_set_title (GTK_WINDOW(ventanaConsulta), "Desafío"); //Pone título
    gtk_box_pack_start (GTK_BOX(contenedorVertical), etiquetaDesafio, FALSE, FALSE, 3);
    gtk_box_pack_start (GTK_BOX(contenedorVertical), etiquetaPregunta, FALSE, FALSE, 3);
    gtk_box_pack_start (GTK_BOX(contenedorVertical), contenedorHorizontal, FALSE, FALSE, 3);
    gtk_box_pack_start (GTK_BOX(contenedorHorizontal), botonSi, FALSE, FALSE, 3);
    gtk_box_pack_start (GTK_BOX(contenedorHorizontal), botonNo, FALSE, FALSE, 3);

    datosDesafio* datos = new datosDesafio();
    datos->ventanaElección = ventanaConsulta;
    datos->pthis = pthis;
    datos->rival = rival;

    g_signal_connect(G_OBJECT(botonNo), "clicked", G_CALLBACK(rechazarBatalla), datos);
    g_signal_connect(G_OBJECT(botonSi), "clicked", G_CALLBACK(aceptarBatalla), datos);

    gtk_container_add(GTK_CONTAINER(ventanaConsulta), contenedorVertical);

    gtk_window_set_transient_for (GTK_WINDOW(ventanaConsulta), GTK_WINDOW(pthis->ventana));
    gtk_window_set_position(GTK_WINDOW(ventanaConsulta), GTK_WIN_POS_CENTER_ON_PARENT);

    gtk_widget_show_all(ventanaConsulta);
}

void InterfazGrafica::analizarMensajesRecibidos(InterfazGrafica* p)
{
    // actúa en función al tipo de mensaje que llegó
    string mensaje, leyenda;
    while (p->contacto.obtenerMensajeRecibido(mensaje))
    {
        switch (mensaje[0])
        {
            case 'C': //chat
                p->agregarMensajeAMostrar(mensaje.substr(1,mensaje.length()-1));
                break;
            case 'D': //se desconectó el servidor
                gtk_label_set_text(GTK_LABEL(p->etiqueta), "No está conectado a ningún
servidor");
                p->agregarMensajeAMostrar(" Se ha desconectado del servidor");
                break;
            case 'U': //usuarios disponibles
        }
    }
}
```

```
        p->adversarios = mensaje.substr(1,mensaje.length()-1);
        p->adversariosActualizados = true;
        break;
    case 'S': // señal del servidor
        p->agregarMensajeAMostrar(mensaje.substr(1,mensaje.length()-1));
        break;
    case 'B': // propuesta de batalla
        p->propuestaBatalla(p,mensaje.substr(1,mensaje.length()-1));
        p->agregarMensajeAMostrar(" Recibe desafío de " + mensaje.substr
(1,mensaje.length()-1));
        break;
    case 'R': // rechazo de batalla
        leyenda = "No podrá jugar con ";
        leyenda += mensaje.substr(1,mensaje.length()-1);
        mostrarAlerta(leyenda,p->ventana);
        p->agregarMensajeAMostrar(" No pudo efectuarse desafío");
        break;
    case 'Z': // inicio de batalla
        iniciarBatalla(p);
        break;
    case 'I': // resultado de una batalla
        leyenda = mensaje.substr(1,mensaje.length()-1);
        mostrarAlerta(leyenda,p->ventana);
        break;
    case 'G': // tabla de posiciones
        leyenda = mensaje.substr(1,mensaje.length()-1);
        mostrarAlerta(leyenda,p->ventana);
        break;
    case 'O': //servidor ocupado
        mostrarAlerta("El Servidor está manejando su máximo \n de partidas
simultáneas. Intente un rato más tarde",p->ventana);
        break;
    }
}
}

void InterfazGrafica::modificar_archivo_gravedad(GtkWidget *widget, gpointer data)
{
    datosGravedad* datos = (datosGravedad*) data;
    string buffer,archivo;
    ifstream arch("config.txt");//,ios::in|ios::out);

    double valorNro = gtk_range_get_value(GTK_RANGE(datos->escala));
    valorNro /= (-100.0);

    std::ostringstream os;
    os << valorNro;
    string valor = os.str();

    getline(arch,buffer);

    archivo = buffer.replace(buffer.find_last_of(" ") + 1, buffer.length() - buffer.find_last_of(" ")
) - 1, valor.c_str(), valor.length());

    archivo += "\n";

    while (getline(arch,buffer))
    {
        archivo += buffer;
        archivo += "\n";
        buffer.clear();
    }

    arch.close();

    // reescritura del archivo
    ofstream arch2("config.txt");
    arch2.write(archivo.c_str(), archivo.length());
    arch2.close();

    gtk_widget_destroy(datos->ventanaGravedad);
}
```

```
    delete datos;
}

void InterfazGrafica::pedir_posiciones(GtkWidget *widget, gpointer data)
{
    InterfazGrafica* pthis = (InterfazGrafica*) data;
    pthis->contacto.enviarMensaje("G");
}

void InterfazGrafica::modificar_gravedad(GtkWidget *widget, gpointer data)
{
    float inicial = 0.0;
    string linea;
    ifstream arch("config.txt");
    getline(arch,linea);
    linea = linea.substr(linea.find_last_of(" ")+1,linea.length()-linea.find_last_of(" ")-1);
    arch.close();
    linea.replace(linea.find("."),1,",",1);
    inicial = strtod(linea.c_str(), NULL);
    InterfazGrafica* pthis = (InterfazGrafica*) data;
    GtkWidget* ventanaGravedad = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_window_set_transient_for (GTK_WINDOW(ventanaGravedad),GTK_WINDOW(pthis->ventana));
    gtk_window_set_position(GTK_WINDOW(ventanaGravedad),GTK_WIN_POS_CENTER_ON_PARENT);
    GtkWidget* escala = gtk_hscale_new(GTK_ADJUSTMENT(gtk_adjustment_new(inicial*(-100), 1.0,
31.0, 1.0, 1.0, 1.0 )));
    gtk_scale_set_draw_value( GTK_SCALE(escala),FALSE );
    gtk_window_set_default_size (GTK_WINDOW (ventanaGravedad), 200, 50);

    GtkWidget* botonOk = gtk_button_new_from_stock("Aceptar");
    GtkWidget* botonCancelar = gtk_button_new_from_stock("Cancelar");
    GtkWidget* contenedorHorizontal = gtk_hbox_new(FALSE, 10);
    GtkWidget* contenedorVertical = gtk_vbox_new(FALSE, 10);

    gtk_window_set_title (GTK_WINDOW (ventanaGravedad), "Gravedad"); //Pone título

    gtk_box_pack_start (GTK_BOX(contenedorVertical), escala, FALSE, FALSE, 10);
    gtk_box_pack_start (GTK_BOX(contenedorVertical), contenedorHorizontal, FALSE, FALSE, 7);
    gtk_box_pack_start (GTK_BOX(contenedorHorizontal), botonOk, FALSE, FALSE, 10);
    gtk_box_pack_start (GTK_BOX(contenedorHorizontal), botonCancelar, FALSE, FALSE, 7);

    gtk_container_add(GTK_CONTAINER(ventanaGravedad), contenedorVertical);

    datosGravedad* datos = new datosGravedad();
    datos->ventanaGravedad = ventanaGravedad;
    datos->escala = escala;

    g_signal_connect(G_OBJECT(botonOk), "clicked", G_CALLBACK(modificar_archivo_gravedad), datos);
    g_signal_connect(G_OBJECT(botonCancelar), "clicked", G_CALLBACK(cerrar_ventana),
ventanaGravedad);

    gtk_widget_show_all (ventanaGravedad);
}

void InterfazGrafica::resultado_juego_solo(InterfazGrafica* pthis, const std::string& puntos)
{
    //Muestro ventana con resultado
    GtkWidget* ventanaResultado = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_window_set_transient_for (GTK_WINDOW(ventanaResultado),GTK_WINDOW(pthis->ventana));
    gtk_window_set_position(GTK_WINDOW(ventanaResultado),GTK_WIN_POS_CENTER_ON_PARENT);
    gtk_window_set_title (GTK_WINDOW (ventanaResultado), "Resultado");

    GtkWidget* contenedorVertical = gtk_vbox_new(FALSE, 10);
    GtkWidget* contenedorHorizontal = gtk_hbox_new(FALSE, 10);

    GtkWidget* botonContinuar = gtk_button_new_from_stock("Continuar");
    GtkWidget* botonTerminar = gtk_button_new_from_stock("Terminar");

    string leyendaEtiqueta = "Ha completado el nivel con ";
}
```

```
leyendaEtiqueta += puntos;
leyendaEtiqueta += " puntos";
GtkWidget* label = gtk_label_new(leyendaEtiqueta.c_str());

gtk_box_pack_start (GTK_BOX(contenedorVertical), label, TRUE, TRUE, 10);
gtk_box_pack_start (GTK_BOX(contenedorHorizontal), botonContinuar, FALSE, FALSE, 10);
gtk_box_pack_start (GTK_BOX(contenedorHorizontal), botonTerminar, FALSE, TRUE, 10);
gtk_box_pack_start (GTK_BOX(contenedorVertical), contenedorHorizontal, FALSE, TRUE, 10);

gtk_signal_connect(GTK_OBJECT(botonContinuar), "clicked", GTK_SIGNAL_FUNC(jugar_solo), pthis);
gtk_signal_connect_object (GTK_OBJECT (botonContinuar), "clicked", GTK_SIGNAL_FUNC
	gtk_widget_destroy), GTK_OBJECT (ventanaResultado));

gtk_signal_connect_object (GTK_OBJECT (botonTerminar), "clicked", GTK_SIGNAL_FUNC
(GTK_WIDGET_DESTROY), GTK_OBJECT (ventanaResultado));

gtk_container_add(GTK_CONTAINER(ventanaResultado), contenedorVertical);

gtk_widget_show_all (ventanaResultado);

}

gboolean InterfazGrafica::timeout_handler(gpointer data)
{

GtkTextBuffer *buffer;
GtkTextIter iter;
InterfazGrafica* pthis = (InterfazGrafica*) data;
string mensaje;

if (pthis->batalla == NULL)
{
    if (pthis->juegoSolo != NULL)
    {
        if (!pthis->juegoSolo->estaJugando())
        {
            // pide los puntos y los manda
            int puntos = pthis->juegoSolo->getPuntos();
            if (puntos > 0)
            {
                enteroAString(mensaje,puntos);
                pthis->agregarMensajeAMostrar(" Ha obtenido " + mensaje + " "
puntos en un juego solitario");
                mensaje = "P" + mensaje;
                pthis->contacto.enviarMensaje(mensaje);
            }
            else
                pthis->contacto.enviarMensaje("PO");
            if (puntos == 0)
                mostrarAlerta("No ha completado correctamente el
escenario",pthis->ventana);

                delete pthis->juegoSolo;
                pthis->juegoSolo = NULL;

                if (puntos > 0)
                    resultado_juego_solo(pthis,mensaje.substr(1,mensaje.length() -
1));
            }
        }
    analizarMensajesRecibidos(pthis);

    //Muestra los mensajes en la pantalla del chat
    pthis->bloquearEstado();
}
```

```

while (!pthis->mensajesAMostrar.empty())
{
    mensaje = pthis->mensajesAMostrar[0];
    pthis->mensajesAMostrar.erase(pthis->mensajesAMostrar.begin());

    //Agrega el mensaje al buffer del cuadro de mensajes

    buffer = gtk_text_view_get_buffer (GTK_TEXT_VIEW (pthis->cuadroMensajes));
    mensaje += "\n";

    gtk_text_buffer_get_end_iter (buffer, &iter);
    gtk_text_buffer_insert (buffer, &iter, mensaje.c_str(), -1);
    gtk_text_buffer_get_end_iter (buffer, &iter);

    gtk_text_view_scroll_to_iter (GTK_TEXT_VIEW (pthis->cuadroMensajes), &iter,
0.0, FALSE, 0, 0);
}
pthis->desbloquearEstado();

}

else
{
    if (!pthis->batalla->estaJugando())
    {

        // pide los puntos y los manda
        int puntos = pthis->batalla->getPuntos();
        if (puntos > 0)
        {
            enteroAString(mensaje,puntos);
            pthis->agregarMensajeAMostrar(" Ha obtenido " + mensaje + " puntos en
un juego doble");
            mensaje = "P" + mensaje;
            pthis->contacto.enviarMensaje(mensaje);
        }
        else
            pthis->contacto.enviarMensaje("P0");

        delete pthis->batalla;
        delete pthis->chat;

        pthis->batalla = NULL;
        pthis->chat = NULL;

    }
    else
        //se tiene que hacer cargo la interfaz batalla
        if (!pthis->batalla->analizarMensajesRecibidos())
        {
            mostrarAlerta("Se ha desconectado del servidor",pthis->ventana);
            gtk_label_set_text(GTK_LABEL(pthis->etiqueta),"No está conectado a
ningún servidor");
            pthis->agregarMensajeAMostrar(" Se ha desconectado del servidor");
        }
    }

    return true;
}

gboolean InterfazGrafica::inicio(gpointer data)
{
    InterfazGrafica * pthis = (InterfazGrafica*) data;
    sleep(1);
    gtk_widget_destroy(pthis->window); //quita presentación

    //inicia ventana principal
}

```

```
gtk_widget_show_all (pthis->ventana);

//agrega función que se ejecuta cada cierto intervalo de tiempo
g_timeout_add(100, timeout_handler, pthis);

return false; //se ejecuta una sola vez
}

void InterfazGrafica::run()
{

    //carga imagen de presentación
    GdkPixbuf *pix = gdk_pixbuf_new_from_file_at_size("ima/armadillo.jpg",100,100,NULL);

    GtkWidget *imagen;
    GdkPixbuf *pixbuf;
    GdkPixmap *pixmap;
    GdkBitmap *mask;

    pixbuf = gdk_pixbuf_new_from_file_at_size("ima/armadillo.jpg",600,650,NULL);
    gdk_pixbuf_render_pixmap_and_mask (pixbuf, &pixmap, &mask, 0);
    imagen = gtk_pixmap_new( pixmap, mask );

    //ventana de presentación
    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title(GTK_WINDOW(window), "TatuCarreta Run");
    gtk_window_set_decorated(GTK_WINDOW(window), FALSE);
    gtk_widget_show(imagen);
    gtk_container_add (GTK_CONTAINER (window), imagen);
    gtk_window_set_position(GTK_WINDOW(window),GTK_WIN_POS_CENTER);
    gtk_widget_show_all (window);

    //inicia entidad de contacto con los servidores
    contacto.start();

    GtkWidget* contenedorHorizontalChat;
    GtkWidget* contenedorVerticalChat;
    GtkWidget* contenedorHorizontalGrande;
    GtkWidget* contenedorVerticalBotones;
    GtkWidget* botonAceptar;

    GtkWidget* scroll;
    GtkWidget* botonConectar;
    GtkWidget* botonJugar;
    GtkWidget* botonJugarCon;
    GtkWidget* botonGravedad;
    GtkWidget* botonDesconectar;
    GtkWidget* botonRanking;

    gtk_set_locale(); //Configura idioma
    ****
    /** Configura la ventana ***/
    ***

    ventana = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_window_set_position(GTK_WINDOW(ventana),GTK_WIN_POS_CENTER);
    gtk_window_set_icon(GTK_WINDOW(ventana),GDK_PIXBUF(pix));
    //Conecta la función on_delete_event a la señal delete_event
    g_signal_connect(G_OBJECT(ventana), "delete_event", G_CALLBACK(on_delete_event), (gpointer)
this );
    //Conecta la función destruir a la señal destroy
    g_signal_connect(G_OBJECT(ventana), "destroy", G_CALLBACK(destruir), NULL);
    gtk_container_set_border_width(GTK_CONTAINER(ventana), 10); //Configura borde
}
```

```
gtk_window_set_title (GTK_WINDOW (ventana), TITULO_VENTANA); //Pone título
gtk_window_set_default_size (GTK_WINDOW (ventana), 500, 500); //Configura tamaño de la ventana

/****************/
/** Genera los contenedores ***/
/****************/

contenedorHorizontalChat = gtk_hbox_new(FALSE, 10);
contenedorVerticalChat = gtk_vbox_new(FALSE, 10);
contenedorHorizontalGrande = gtk_hbox_new(FALSE, 10);
contenedorVerticalBotones = gtk_vbox_new(FALSE, 10);

/****************/
/** Genera la etiqueta ***/
/****************/

etiqueta = gtk_label_new("No está conectado a ningún servidor");

//Agredo la etiqueta al contenedor vertical de chat
gtk_box_pack_start (GTK_BOX(contenedorVerticalChat), etiqueta, FALSE, FALSE, 3);

/****************/
/** Junto los contenedores ***/
/****************/

//Coloca en la ventana al contenedor principal
gtk_container_add(GTK_CONTAINER(ventana), contenedorHorizontalGrande);

//Agredo el contenedor de chat en el principal
gtk_box_pack_start (GTK_BOX(contenedorHorizontalGrande), contenedorVerticalChat, TRUE, TRUE,
3);

//Agredo el contenedor de botones en el principal
gtk_box_pack_start (GTK_BOX(contenedorHorizontalGrande), contenedorVerticalBotones, FALSE,
FALSE, 3);

/****************/
/** Genera el área de texto donde se muestran los mensajes ***/
/****************/

cuadroMensajes = gtk_text_view_new ();
gtk_widget_set_usize(cuadroMensajes, 50, -2); //Configura tamaño
gtk_text_view_set_editable(GTK_TEXT_VIEW(cuadroMensajes), FALSE); //Evita que sea texto
editable
gtk_text_view_set_cursor_visible(GTK_TEXT_VIEW(cuadroMensajes), FALSE);
gtk_text_view_set_pixels_above_lines(GTK_TEXT_VIEW(cuadroMensajes), 5);
gtk_text_view_set_left_margin(GTK_TEXT_VIEW(cuadroMensajes), 3);

/****************/
/** Genera el scroll para el espacio de texto ***/
/****************/

scroll = gtk_scrolled_window_new(NULL,NULL);
gtk_scrolled_window_add_with_viewport(GTK_SCROLLED_WINDOW(scroll),cuadroMensajes); //Agrega el
texto
gtk_box_pack_start (GTK_BOX(contenedorVerticalChat), scroll, TRUE, TRUE, 3); //Agrega el
scroll al contenedor vertical
gtk_scrolled_window_set_policy (GTK_SCROLLED_WINDOW (scroll), GTK_POLICY_AUTOMATIC,
GTK_POLICY_ALWAYS); //Hace automático el scroll horizontal

//Agredo el contenedor horizontal del chat en el vertical
gtk_box_pack_start (GTK_BOX(contenedorVerticalChat), contenedorHorizontalChat, FALSE, TRUE, 3);

/****************/
/** Genera el espacio de entrada de texto ***/
/****************/
```

```
*****  
entrada = gtk_entry_new();  
gtk_entry_set_width_chars(GTK_ENTRY(entrada), 50); //Configura tamaño  
  
//Agrega la entrada de texto al contenedor horizontal del chat  
gtk_box_pack_start (GTK_BOX(contenedorHorizontalChat), entrada, TRUE, TRUE, 3);  
  
*****  
/** Genera label auxiliar **/  
*****  
  
GtkWidget* labelAuxiliar = gtk_label_new("\n\n\n");  
  
//Agrego la etiqueta al contenedor de botones  
gtk_box_pack_start (GTK_BOX(contenedorVerticalBotones), labelAuxiliar, FALSE, FALSE, 3);  
  
*****  
/** Genera el botón del chat **/  
*****  
  
botonAceptar = gtk_button_new_from_stock("gtk-ok");  
  
//Agrego el botón al contenedor horizontal  
gtk_box_pack_start (GTK_BOX(contenedorHorizontalChat), botonAceptar, FALSE, FALSE, 3);  
  
*****  
/** Genera el botón de conectar automatico **/  
*****  
/* Botón para agilizar las pruebas. No se ve actualmente en la pantalla */  
  
GtkWidget * botonConectarAuto = gtk_button_new_from_stock("ConectarAuto");  
  
//Agrego el botón al contenedor de botones  
gtk_box_pack_start (GTK_BOX(contenedorVerticalBotones), botonConectarAuto, FALSE, FALSE, 6);  
  
//  
g_signal_connect(G_OBJECT(botonConectarAuto), "clicked", G_CALLBACK  
(apretar_boton_conectar_auto), this);  
  
*****  
/** Genera el botón de conectar **/  
*****  
  
botonConectar = gtk_button_new_from_stock("Conectar");  
  
//Agrego el botón al contenedor de botones  
gtk_box_pack_start (GTK_BOX(contenedorVerticalBotones), botonConectar, FALSE, FALSE, 6);  
  
*****  
/** Genera el botón de desconectar **/  
*****  
  
botonDesconectar = gtk_button_new_from_stock("Desconectar");  
  
//Agrego el botón al contenedor de botones  
gtk_box_pack_start (GTK_BOX(contenedorVerticalBotones), botonDesconectar, FALSE, FALSE, 6);  
  
*****  
/** Genera el botón de jugar **/  
*****  
  
botonJugar = gtk_button_new_from_stock("Jugar solo");
```

```
//Agrego el botón al contenedor de botones
gtk_box_pack_start (GTK_BOX(contenedorVerticalBotones), botonJugar, FALSE, FALSE, 6);

/*****************/
/** Genera el botón de jugar con **/
/*****************/

botonJugarCon = gtk_button_new_from_stock("Jugar con...");

//Agrego el botón al contenedor de botones
gtk_box_pack_start (GTK_BOX(contenedorVerticalBotones), botonJugarCon, FALSE, FALSE, 6);

/*****************/
/** Genera el botón de cambiar gravedad **/
/*****************/

botonGravedad = gtk_button_new_from_stock("Set Gravedad");

//Agrego el botón al contenedor de botones
gtk_box_pack_start (GTK_BOX(contenedorVerticalBotones), botonGravedad, FALSE, FALSE, 6);

/*****************/
/** Genera el botón de ranking *****/
/*****************/

botonRanking = gtk_button_new_from_stock("Posiciones");

//Agrego el botón al contenedor de botones
gtk_box_pack_start (GTK_BOX(contenedorVerticalBotones), botonRanking, FALSE, FALSE, 6);

//Conexión de señales

g_signal_connect(G_OBJECT(botonConectar), "clicked", G_CALLBACK(apretar_boton_conectar), this);
g_signal_connect(G_OBJECT(botonAceptar), "clicked", G_CALLBACK(apretar_boton_aceptar), this);
gtk_signal_connect (GTK_OBJECT(entrada), "activate", GTK_SIGNAL_FUNC(apretar_boton_aceptar),
this);

g_signal_connect(G_OBJECT(botonDesconectar), "clicked", G_CALLBACK(desconectar), this);
g_signal_connect(G_OBJECT(botonJugar), "clicked", G_CALLBACK(jugar_solo), this);
g_signal_connect(G_OBJECT(botonJugarCon), "clicked", G_CALLBACK(jugar_batalla), this);
g_signal_connect(G_OBJECT(botonGravedad), "clicked", G_CALLBACK(modificar_gravedad), this);
g_signal_connect(G_OBJECT(botonRanking), "clicked", G_CALLBACK(pedir_posiciones), this);

// Función que cierra la presentación
g_timeout_add(1000, inicio, this);

gtk_main();

contacto.terminar();
contacto.join();

}

bool InterfazGrafica::obtenerMensajeLeido(string& mensaje)
{
    bool retorno;
    bloquearEstado();
    if (!mensajesLeidos.empty())
```

```
{  
    retorno = true;  
    mensaje = mensajesLeidos[0]; //Da el primer mensaje leido disponible  
    mensajesLeidos.erase(mensajesLeidos.begin()); //Lo saca del vector  
}  
else  
    retorno = false;  
desbloquearEstado();  
return retorno;  
}  
  
void InterfazGrafica::agregarMensajeAMostrar(const std::string& mensaje)  
{  
    bloquearEstado();  
    mensajesAMostrar.push_back(mensaje);  
    desbloquearEstado();  
}  
  
void InterfazGrafica::enteroAString(std::string& cadena, ulong entero)  
{  
    if (entero == 0)  
        cadena = "0";  
    else  
        cadena = "";  
    while (entero != 0)  
    {  
        switch (entero % 10)  
        {  
            case 0: cadena="0"+cadena; break;  
            case 1: cadena="1"+cadena; break;  
            case 2: cadena="2"+cadena; break;  
            case 3: cadena="3"+cadena; break;  
            case 4: cadena="4"+cadena; break;  
            case 5: cadena="5"+cadena; break;  
            case 6: cadena="6"+cadena; break;  
            case 7: cadena="7"+cadena; break;  
            case 8: cadena="8"+cadena; break;  
            case 9: cadena="9"+cadena; break;  
        }  
        entero /= 10;  
    }  
}
```

```
#include "client_InterfazJugarSolo.h"
#include "InterfazEdicion.h"

InterfazJugarSolo::InterfazJugarSolo(Mapa* mapa)
{
    this->mapa = mapa;
    jugando = true;
    puntos = 0;
}

bool InterfazJugarSolo::estaJugando()
{
    bool ret;
    bloquearEstado();
    ret = jugando;
    desbloquearEstado();
    return ret;
}

int InterfazJugarSolo::getPuntos()
{
    int ret;
    bloquearEstado();
    ret = puntos;
    desbloquearEstado();
    return ret;
}

void InterfazJugarSolo::setJugando(bool valor)
{
    bloquearEstado();
    jugando = valor;
    desbloquearEstado();
}

void InterfazJugarSolo::run()
{
    InterfazEdicion interfaz(mapa);
    interfaz.comenzar(NULL);

    //Guardo los puntos que hizo el jugador
    bloquearEstado();
    //set puntos
    puntos = mapa->getPuntos();
    desbloquearEstado();

    setJugando(false);
}
```

```
#include "common_SocketCliente.h"
#include <iostream>
using namespace std;

#include <glib.h>
#include <glib/gprintf.h>
#include <gtk/gtk.h>
#include "client_InterfazGrafica.h"

#define TITULO_VENTANA "TatuCarreta Run"

int main(int argc, char* argv[])
{
    gtk_init(&argc, &argv);
    InterfazGrafica inter;
    inter.start();

    //gtk_main();
    inter.join();

    /*SocketCliente socket;
    string mensaje = argv[2];
    socket.create();
    if (!socket.connect("127.0.0.1",atoi(argv[1]))) cout << "no se conecto" << endl;
    if (!socket.send(mensaje)) cout << "no envio" << endl;
    socket.recv(mensaje);
    cout << mensaje << endl;
    cin >> mensaje;
    while (mensaje.compare("salir") != 0)
    {
        socket.send(mensaje);
        cin >> mensaje;
    }

    socket.send("D");
    socket.close(); */
}
```

```
#include "Cohete.h"

Cohete::Cohete() :
    ElementoLongitudinal("Longitudinal:Cohete", Coordenada(long(10), long(10)), 25, 0, 10) {}

Cohete::Cohete(Cordenada posicion, ulong magnitud, long angulo, long ancho) :
    ElementoLongitudinal("Longitudinal:Cohete", posicion, magnitud, angulo,
                           ancho) {}

Cohete::~Cohete() {
```

```
#include "common_Mutex.h"

Mutex::Mutex()
{
    pthread_mutex_init(&mut, NULL);
}

void Mutex::lock()
{
    pthread_mutex_lock(&mut);
}

void Mutex::unlock()
{
    pthread_mutex_unlock(&mut);
}
```

```
#include "common_Socket.h"
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <iostream>
using namespace std;

Socket::Socket()
{
    //Aún no está creado
    id = SOCKET_INVALIDO;

    //Inicializa dirección y buffer de remanentes
    memset(&(dir.sin_zero), '\0', 8);
    memset(historial, 0, TAM_BUFFER + 1 );
    validosHistorial = 0;
}

Socket::~Socket()
{
    if (id != SOCKET_INVALIDO)
        ::close(id); //Cierra Socket
}

bool Socket::create()
{
    //Crea Socket
    id = socket(AF_INET, SOCK_STREAM, 0);

    //Analiza resultado
    if (id == SOCKET_INVALIDO)
        return false;
    return true;
}

bool Socket::send(const string& dato) const
{
    unsigned int enviados = 0;
    int retorno;
    string aux = dato;

    /* Envía hasta que la cantidad de enviados sea el tamaño del dato más el \0
     * que indica fin de mensaje. */
    while (enviados != (dato.size() + 1))
    {
        retorno = ::send (id,aux.c_str(),dato.size() + 1 - enviados, MSG_NOSIGNAL);
        if (retorno == -1)
            return false;
        enviados += retorno; //Suma los enviados.
        if (enviados != (dato.size() + 1))
            aux = dato.substr(enviados); //Carga en aux los datos que faltan enviar
    }
    return true;
}

bool Socket::recv(string& dato)
{
    char buffer [TAM_BUFFER + 1];
    bool listo = false;
    dato = "";
    int retorno;
    unsigned int recibidos;

    //Analiza si hay elementos remanentes que son parte del mensaje actual.
    if (validosHistorial != 0)
    {
```

```
dato = historial; //Toma hasta el primer \0
    if (dato.size() != validosHistorial) //Si el dato no tomó todos los remanentes es
porque encontró un \0
    {
        listo = true;
        validosHistorial -= (dato.size()+1); //Resto el tamaño del mensaje mas el \0
    }
else
    validosHistorial = 0;

//Coloca en historial los nuevos remanentes.
memcpy(buffer,historial+dato.size()+1,validosHistorial);
memset(historial, 0, TAM_BUFFER + 1 );
memcpy(historial,buffer,validosHistorial);

}

//Inicializa el buffer
memset (buffer,0,TAM_BUFFER + 1 );

//Recibe hasta encontrar el \0
while (!listo)
{
    retorno = ::recv(id,buffer,TAM_BUFFER, 0 );
    if ((retorno == -1) || (retorno == 0))
        return false;
    recibidos = retorno;
    if (strlen(buffer) < recibidos)
    {
        listo = true;
        if (strlen(buffer) < recibidos - 1)
            //Tengo que copiar a los remanentes
            validosHistorial = recibidos-(strlen(buffer)+1);
            memcpy(historial,buffer+strlen(buffer)+1,validosHistorial);
    }
    dato += buffer; //Concatena lo leído al dato.
    memset(buffer, 0, TAM_BUFFER + 1 );
}
return true;
}

void Socket::setId(const int id)
{
    this->id = id;
}

int Socket::getId() const
{
    return id;
}

sockaddr_in& Socket::getDir()
{
    return dir;
}

void Socket::close()
{
    if (id == SOCKET_INVALIDO)
        return;
    ::close(id);
    id = SOCKET_INVALIDO;
}

void Socket::shutdown(int modo)
{
    if (id == SOCKET_INVALIDO)
        return;
    ::shutdown(id,modo);
    id = SOCKET_INVALIDO;
```

}

```
#include "common_SocketCliente.h"
#include <iostream>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <string>
using namespace std;

bool SocketCliente::connect(const string direccion, const int puerto)
{
    int id = getId();
    sockaddr_in& dir = getDir();

    if (id == SOCKET_INVALIDO)
    {
        if (!this->create())
            return false;
        id = this->getId();
    }

    //Prepara la conexión
    struct hostent *he;
    he = gethostbyname(direccion.c_str());

    if (he == NULL)
        return false;

    dir.sin_family = AF_INET;
    dir.sin_port = htons (puerto);
    dir.sin_addr = *((struct in_addr *)he->h_addr);

    //Intenta conectarse.
    if (::connect(id,(sockaddr*)&dir,sizeof(dir)) == -1)
        return false;

    return true;
}
```

```
#include "common_Thread.h"

#include <signal.h>
#include <unistd.h>

Thread::Thread()
{
    vivo = true;
}

void Thread::start()
{
    corriendo = true;
    pthread_create(&hilo,NULL,static_run,this);
}

Thread::~Thread()
{
    if (corriendo)
    {
        vivo = false; //Indica al hilo que debe suicidarse.
        this->join();
    }
}

void Thread::join() const
{
    pthread_join(hilo,NULL);
}

bool Thread::estaVivo()
{
    bool retorno;
    bloquearEstado();
    retorno = vivo;
    desbloquearEstado();
    return retorno;
}

void* Thread::static_run (void* p)
{
    Thread* pthis = (Thread*) p;
    pthis->run();
    return NULL;
}

void Thread::terminar()
{
    bloquearEstado();
    vivo = false;
    desbloquearEstado();
}

bool Thread::estaCorriendo()
{
    bool retorno;
    bloquearEstado();
    retorno = corriendo;
    desbloquearEstado();
    return retorno;
}

void Thread::bloquearEstado()
```

```
mHilo.lock();  
}  
  
void Thread::desbloquearEstado()  
{  
    mHilo.unlock();  
}  
  
void Thread::setCorriendo(bool valor)  
{  
    corriendo = valor;  
}
```

```
#include "config_file.h"
#include <fstream>

namespace
{
    void readConfigData(std::map<std::string, std::string>& data,
                        std::ifstream& configFile)
    {
        std::string line;

        while (std::getline(configFile, line))
        {
            if (line[0] == '#')
                continue;

            size_t eqPos = line.find('=');
            if (eqPos == linenpos)
                continue;
            std::string key = line.substr(0, eqPos);
            std::string value = line.substr(eqPos + 1);

            data[key] = value;
        }
    }

ConfigFile::ConfigFile(const std::string& filename)
{
    std::ifstream configFile(filename.c_str());

    if (configFile.is_open())
        readConfigData(data_, configFile);
}

ConfigFile::~ConfigFile()
{
}

std::string ConfigFile::readString(const std::string& key) const
{
    TStrStrMap::const_iterator it = data_.find(key);
    if (it == data_.end())
        return std::string();
    else
        return it->second;
}
```

```
#include "Coordenada.h"
#include <math.h>
#include <iostream>
#include "constantes.h"

using namespace std;

Coordenada::Coordenada() {

}

Coordenada::Coordenada(long x, long y) {
    this->x = x;
    this->y = y;
    actualizarAngulo();
    actualizarModulo();
}

Coordenada::Coordenada(ulong modulo, long angulo) {
    this->modulo = modulo;
    this->angulo = angulo;
    actualizarX();
    actualizarY();
}

Coordenada::~Coordenada() {
}

void Coordenada::modificarDelta(Cordenada delta) {
    x += delta.x;
    y += delta.y;
}

void Coordenada::modificar(Cordenada nueva) {
    x = nueva.x;
    y = nueva.y;
}

void Coordenada::imprimir() const {
    cout << "< " << this->x << " , " << this->y << " > " << endl;
    /*
     << "Angulo: "
         << angulo
         << " Modulo: "
         << modulo << endl;
    */
}

void Coordenada::setX(long x) {
    this->x = x;
    actualizarAngulo();
    actualizarModulo();
}

void Coordenada::setY(long y) {
    this->y = y;
    actualizarAngulo();
    actualizarModulo();
}

void Coordenada::setAngulo(long angulo) {
    this->angulo = angulo;
    actualizarX();
    actualizarY();
}

void Coordenada::setModulo(ulong modulo) {
    this->modulo = modulo;
    actualizarX();
    actualizarY();
}
```

```
Coordenada Coordenada::operator+(const Coordenada c) const {
    Coordenada aux = Coordenada(this->x + c.x, this->y + c.y);
    return aux;
}

Coordenada Coordenada::operator-(const Coordenada c) const {
    Coordenada aux = Coordenada(this->x - c.x, this->y - c.y);
    return aux;
}

void Coordenada::actualizarModulo() {
    modulo = sqrt(pow(x, 2) + pow(y, 2));
}

void Coordenada::actualizarAngulo() {
    angulo = Angulo::aGrados(atan2(y,x));
    Angulo::corregir(&angulo);
}

void Coordenada::actualizarX() {
    this->x = modulo * cos(Angulo::aRadianes(angulo));
}

void Coordenada::actualizarY() {
    y = modulo * sin(Angulo::aRadianes(angulo));
}
```

```
//cronometro.cpp

#include "cronometro.h"
int Cronometro::estaParado() { return parado; }

void Cronometro::iniciar() {
    comienzo=clock();
    parado=0;
}

void Cronometro::detener() {
    final=clock();
    parado=1;
}

long Cronometro::getMilisegundos() {
    if (estaParado()) return ((final-comienzo)*1000.0)/CLOCKS_PER_SEC;
    return ((clock()-comienzo)*1000.0)/CLOCKS_PER_SEC;
}
```

```
#include "element.h"

Element::Element(unsigned f, unsigned l){
    first = f;
    last = l;
}

unsigned Element::getFirst(){
    return masas.at(0);
}

unsigned Element::getLast(){
    return masas.at(masas.size()-1);
}
```

```
#include "elementManager.h"

ElementManager::ElementManager(){
    vectoresElementos.push_back(&ropes);
    vectoresElementos.push_back(&cohetes);
    vectoresElementos.push_back(&ruedas);
    vectoresElementos.push_back(&plataformas);
    vectoresElementos.push_back(&metalBars);
    vectoresElementos.push_back(&mainBalls);
    vectoresElementos.push_back(&lonas);
}

ElementManager::~ElementManager(){
    for (size_t i=0; i< vectoresElementos.size(); i++)
        for (size_t j=0; j< vectoresElementos.at(i)->size(); j++)
            delete(vectoresElementos.at(i)->at(j));

    for (size_t i=0; i< zonasLlegada.size(); i++)
        delete(zonasLlegada.at(i));
}

void ElementManager::addRope(int first, int last){
    Element* elemento = new Element(first,last);
    for(int i=first; i<=last;i++)
        elemento->masas.push_back(i);

    ropes.push_back(elemento);
}

void ElementManager::addCohete(int first, int last){
    Element* elemento = new Element(first,last);
    for(int i=first; i<=last;i++)
        elemento->masas.push_back(i);

    cohetes.push_back(elemento);
}

void ElementManager::addRueda(int first, int last){
    Element* elemento = new Element(first,last);
    for(int i=first; i<=last;i++)
        elemento->masas.push_back(i);

    ruedas.push_back(elemento);
}

void ElementManager::addMetalBar(int first, int last){
    Element* elemento = new Element(first,last);
    for(int i=first; i<=last;i++)
        elemento->masas.push_back(i);

    metalBars.push_back(elemento);
}

void ElementManager::addPlataforma(int first, int last){
    Element* elemento = new Element(first,last);
    for(int i=first; i<=last;i++)
        elemento->masas.push_back(i);

    plataformas.push_back(elemento);
}

void ElementManager::addMainBall(int first, int last){
    Element* elemento = new Element(first,last);
    for(int i=first; i<=last;i++)
        elemento->masas.push_back(i);

    mainBalls.push_back(elemento);
}

void ElementManager::addLona(int first, int last){
```

```
Element* elemento = new Element(first,last);
for(int i=first; i<=last;i++)
    elemento->masas.push_back(i);

lonas.push_back(elemento);
}

void ElementManager::addPuntoFijo(int first, int last){
    Element* elemento = new Element(first,last);
    elemento->masas.push_back(first);
    puntosFijos.push_back(elemento);
}

void ElementManager::addZonaLlegada(double x,double y,double width,double heigth){
    zonasLlegada.push_back(new ZonaLlegada(x,y,width,heigth));
}

std::vector<Element*> ElementManager::getRopes(){
    return ropes;
}
std::vector<Element*> ElementManager::getPuntosFijos(){
    return puntosFijos;
}
std::vector<Element*> ElementManager::getRuedas(){
    return ruedas;
}
std::vector<Element*> ElementManager::getMainBalls(){
    return mainBalls;
}
std::vector<Element*> ElementManager::getLonas(){
    return lonas;
}
std::vector<Element*> ElementManager::getCohetes(){
    return cohetes;
}
std::vector<Element*> ElementManager::getPlataformas(){
    return plataformas;
}
std::vector<Element*> ElementManager::getMetalBars(){
    return metalBars;
}
std::vector<ZonaLlegada*> ElementManager::getZonasLlegada(){
    return zonasLlegada;
}
```

```
#include "Elemento.h"
#include "Angulo.h"
#include "math.h"

using namespace std;

Elemento::Elemento() {
    this->bloqueado = false;
}

Elemento::Elemento(const std::string & nombre, Coordenada posicion,
                   const ulong magnitud, long angulo) {
    this->nombre = nombre;
    this->posicion = posicion;
    this->magnitud = magnitud;
    Angulo::corregir(&angulo);
    this->angulo = angulo;
    this->bloqueado = false;
    this->mostrarEnlaces = false;
}

Elemento::~Elemento() {}

void Elemento::setMagnitud(const long valor) {
    if(!bloqueado)
        this->magnitud = valor;
}

ulong Elemento::getMagnitud() const {
    return this->magnitud;
}

void Elemento::setPosicion(const Coordenada nuevaPosicion) {
    if(!bloqueado)
        this->posicion = nuevaPosicion;
}

Coordenada Elemento::getPosicion() const {
    return this->posicion;
}

void Elemento::setAngulo(const long angulo) {
    if(!bloqueado)
        this->angulo = angulo;
}

long Elemento::getAngulo() const {
    return this->angulo;
}

Coordenada Elemento::getExtremoOpuesto() const {
    Coordenada extOp;
    extOp.setX(magnitud * cos(Angulo::aRadianes(angulo)) + posicion.x);
    extOp.setY(magnitud * sin(Angulo::aRadianes(angulo)) + posicion.y);
    return extOp;
}

void Elemento::bloquear() {
    bloqueado = true;
}

void Elemento::desbloquear() {
    bloqueado = false;
}

bool Elemento::estaBloqueado() const {
    return bloqueado;
}

string Elemento::getNombre() const {
    return nombre;
```

```
}

bool Elemento::es(const int nro_elemento) const {
    switch(nro_elemento) {
        case MASA: return (nombre.find("Masa")!=string::npos)? true : false;
        case PUNTO_FIJO: return (nombre.find("PuntoFijo")!=string::npos)? true : false;
        case RUEDA: return (nombre.find("Rueda")!=string::npos)? true : false;
        case BARRA: return (nombre.find("Barra")!=string::npos)? true : false;
        case PLATAFORMA: return (nombre.find("Plataforma")!=string::npos)? true : false;
        case CINTA: return (nombre.find("Cinta")!=string::npos)? true : false;
        case SOGA: return (nombre.find("Soga")!=string::npos)? true : false;
        case COHETE: return (nombre.find("Cohete")!=string::npos)? true : false;
        case CIRCULAR: return (nombre.find("Circular")!=string::npos)? true : false;
        case LONGITUDINAL: return (nombre.find("Longitudinal")!=string::npos)? true : false;
        case ENLAZABLE: return (nombre.find("Enlazable")!=string::npos)? true : false;
        case PUNTO_ENLACE: return (nombre.find("PuntoDeEnlace")!=string::npos)? true : false;
        case ZONA_LLLEGADA: return (nombre.find("ZonaLlegada")!=string::npos)? true : false;
    }
    return false;
}

int Elemento::getTipo() const
{
    string nom = nombre.substr(nombre.find_last_of(";")+1,nombre.length()-nombre.find_last_of(";")-1);
    if (nom.compare("Masa") == 0)
        return MASA;

    if (nom.compare("Rueda") == 0)
        return RUEDA;

    if (nom.compare("PuntoFijo") == 0)
        return PUNTO_FIJO;

    if (nom.compare("Barra") == 0)
        return BARRA;

    if (nom.compare("Plataforma") == 0)
        return PLATAFORMA;

    if (nom.compare("Soga") == 0)
        return SOGA;

    if (nom.compare("Cinta") == 0)
        return CINTA;

    if (nom.compare("Cohete") == 0)
        return COHETE;

    if (nom.compare("ZonaLlegada") == 0)
        return ZONA_LLLEGADA;

    return NINGUN_ELEMENTO;
}

void Elemento::imprimir() {
    cout << "Elemento: " << getNombre() << endl;
    cout << "Posicion actual: ";
    this->posicion.imprimir();
    //cout << "Posicion inicial: ";
    //this->bufferPosicionInicial.imprimir();
    cout << "Magnitud: " << this->magnitud << endl
        << "Angulo actual: " << this->angulo << endl << endl;
}

bool Elemento::getMostrarPuntosDeEnlace() const {
    return mostrarEnlaces;
}
```

```
void Elemento::setMostrarPuntosDeEnlace(const bool b) {
    mostrarEnlaces = b;
}
```

```
#include "ElementoCircular.h"

ElementoCircular::ElementoCircular(const std::string & nombre,
                                    Coordenada posicion, const ulong magnitud, long angulo) :
    Elemento(nombre, posicion, magnitud, angulo) {

}

ElementoCircular::ElementoCircular() {

}

ElementoCircular::~ElementoCircular() {

}

int ElementoCircular::seleccionar(Coordenada punto) const {
    int seleccion= ELEMENTO_NO_SELECCIONADO;
    Coordenada resta = punto - getPosition();

    if (resta.modulo <= getMagnitud()) {
        seleccion = ELEMENTO_SELECCIONADO POR CENTRO;
    }

    return seleccion;
}
```

```
#include "ElementoEnlazable.h"

using namespace std;

ElementoEnlazable::ElementoEnlazable(const std::string & nombre,
                                      Coordenada posicion, const ulong magnitud, long angulo, Elemento * poseedor) :
    ElementoCircular(nombre, posicion, magnitud, angulo) {
    this->poseedor = poseedor;
    enlazado = false;
}

ElementoEnlazable::~ElementoEnlazable() {
}

Elemento * ElementoEnlazable::getPoseedor() const {
    return poseedor;
}

list <ElementoEnlazable*> ElementoEnlazable::getListaEnlaces() const {
    return _enlaces;
}

void ElementoEnlazable::eliminarEnlaces() {
    list <ElementoEnlazable*>::iterator it;
    for(it = _enlaces.begin(); it != _enlaces.end(); it++) {
        delete (*it);
    }
    _enlaces.clear();
}

bool ElementoEnlazable::conectarA(ElementoEnlazable* enganche) {
    if(enganche != this)
        if((this->getPosicion().x == enganche->getPosicion().x) && (this->getPosicion().y ==
enganche->getPosicion().y)) {
            _enlaces.push_back(enganche);
            return true;
        }
    return false;
}
```

```
#include "ElementoLongitudinal.h"
#include <math.h>
#include <iostream>
using namespace std;

ElementoLongitudinal::ElementoLongitudinal() {

}

ElementoLongitudinal::ElementoLongitudinal(const std::string & nombre,
                                             Coordenada posicion, const ulong magnitud, long angulo, ulong ancho) :
    Elemento(nombre, posicion, magnitud, angulo) {
    Coordenada p2;
    p2.setAngulo(angulo);
    p2.setModulo(magnitud);
    p2 = p2 + posicion;
    extremoOpuesto = p2;
    this->ancho = ancho;
}

ElementoLongitudinal::~ElementoLongitudinal() {

}

void ElementoLongitudinal::setPosicion(const Coordenada nuevaPosicion) {
    Elemento::setPosicion(nuevaPosicion);

    Coordenada p2;
    long ang = -getAngulo();

    p2.setX( cos(Angulo::aRadianes(ang))*getMagnitud() + getPosicion().x );
    p2.setY( sin(Angulo::aRadianes(ang))*getMagnitud() + getPosicion().y );
    extremoOpuesto = p2;
}

void ElementoLongitudinal::setAngulo(const long angulo) {
    Elemento::setAngulo(angulo);
}

void ElementoLongitudinal::actualizarPuntosDeEnlace(int modo) {
    int cantidadEnlaces = getMagnitud() / LONGITUD_DE_ENLACE_MINIMA;
    if(modo == 1) cantidadEnlaces = 1;

    int separacionEntreEnlaces = (cantidadEnlaces == 0) ? this->getMagnitud()
                                                       : (getMagnitud() / cantidadEnlaces);

    eliminarEnlaces();
    _enlaces.clear();

    /* Completa la lista de coordenadas de enlace sobre el elemento longitudinal */
    for(ulong paso = 0; paso <= (this->getMagnitud() - separacionEntreEnlaces); paso += separacionEntreEnlaces) {
        Coordenada posicionEnlace;
        posicionEnlace.setModulo(paso);
        posicionEnlace.setAngulo(-this->getAngulo());
        posicionEnlace = posicionEnlace + this->getPosicion();
        PuntoDeEnlace * nuevoEnlace = new PuntoDeEnlace(posicionEnlace, this);
        _enlaces.push_back(nuevoEnlace);
    }
    _enlaces.push_back(new PuntoDeEnlace(this->getExtremoOpuesto(), this));
}

void ElementoLongitudinal::eliminarEnlaces() {
    list <PuntoDeEnlace*>::iterator it;
    for(it = _enlaces.begin(); it != _enlaces.end(); it++) {
        delete (*it);
    }
    _enlaces.clear();
}
```

```
void ElementoLongitudinal::imprimirEnlaces() {
    list <PuntoDeEnlace*>::iterator it;
    for(it = _enlaces.begin(); it != _enlaces.end(); it++) {
        (*it)->getPosicion().imprimir();
    }
    cout << endl << endl;
}

std::list<PuntoDeEnlace*> ElementoLongitudinal::getListaEnlaces() {
    return _enlaces;
}

ulong ElementoLongitudinal::getAncho() const {
    return ancho;
}

void ElementoLongitudinal::setExtremo0puesto(const Coordenada p2) {
    extremo0puesto = p2;

    Coordenada resta = p2 - getPosicion();
    setMagnitud(resta.modulo);

    if (resta.angulo<180)
        resta.angulo=-resta.angulo;

    setAngulo(resta.angulo);
}

Coordenada ElementoLongitudinal::getExtremo0puesto() const {
    return extremo0puesto;
}

int ElementoLongitudinal::seleccionar(Cordenada punto) const {
    Coordenada aux = punto - this->getPosicion(); //traslado el punto al origen

    if (aux.angulo<180)
        aux.angulo=-aux.angulo;

    aux.setAngulo(aux.angulo - this->getAngulo()); //roto

    int resultado= ELEMENTO_NO_SELECCIONADO;

    /* Analisis de si se hizo click y en donde */
    if ( (aux.x <= this->getMagnitud()) && (fabs(aux.y) <= (this->ancho +
MARGEN_DE_AGARRE_ANCHO)) ) {
        resultado = ELEMENTO_SELECCIONADO_POR_CENTRO;
        if (aux.x > (long)(this->getMagnitud() - MARGEN_DE_AGARRE))
            resultado = ELEMENTO_SELECCIONADO_POR_EXTRÉM02;
    }

    return resultado;
}
```

```
#include "FabricaDeElementos.h"

FabricaDeElementos::FabricaDeElementos() {
}

FabricaDeElementos::~FabricaDeElementos() {
}

Elemento * FabricaDeElementos::crear(int nroElemento, Coordenada posicion) {
    switch (nroElemento) {
        case MASA:
            return new Masa(posicion, 20);
        case RUEDA:
            return new Rueda(posicion, 30);
        case PUNTO_FIJO:
            return new PuntoFijo(posicion, 25);
        case BARRA:
            return new Barra(posicion, LONGITUD_MINIMA, 0, 5);
        case PLATAFORMA:
            return new Plataforma(posicion, LONGITUD_MINIMA, 0, 6);
        case SOGA:
            return new Soga(posicion, LONGITUD_MINIMA, 0, 5);
        case CINTA:
            return new Cinta(posicion, LONGITUD_MINIMA, 0, 5);
        case COHETE:
            return new Cohete(posicion, LONGITUD_COHETE , 0, 20);
        case ZONA_LLEGADA:
            return new Llegada(posicion);
    }
    return NULL;
}
```

```
#include "fileutils.h"
#include <SDL.h>

BinFile::BinFile(const char *filename) :
s_(filename, std::ios::in | std::ios::binary)
{
}

BinFile::~BinFile()
{
}

char BinFile::readbyte()
{
    return s_.get();
}

int16_t BinFile::readword()
{
    uint16_t ret;
    s_.read(reinterpret_cast<char*>(&ret), sizeof(ret));
#ifndef SDL_BYTEORDER != SDL_LIL_ENDIAN
    ret = SDL_Swap16(ret);
#endif
    return ret;
}

int32_t BinFile::read dword()
{
    uint32_t ret;
    s_.read(reinterpret_cast<char*>(&ret), sizeof(ret));
#ifndef SDL_BYTEORDER != SDL_LIL_ENDIAN
    ret = SDL_Swap32(ret);
#endif
    return ret;
}

void BinFile::readbytes(char* buffer, size_t num)
{
    s_.read(buffer, num);
}

size_t BinFile::tell()
{
    return s_.tellg();
}

void BinFile::seek(size_t pos)
{
    s_.seekg(pos);
}
```

```
#include "hiloCron.h"
using namespace std;

HiloCron::HiloCron()
{
}

HiloCron::~HiloCron()
{
}

void HiloCron::run()
{
    cronometro = new Cronometro();
    cronometro->iniciar();
    while (estaVivo())
    {
        sleep(0);
    }
    cronometro->detener();
    delete(cronometro);
    cronometro = NULL;
}

long HiloCron::getMilisegundos(){
    if (cronometro) return cronometro->getMilisegundos();
    return -1;
}
```

```
#include "InterfazEdicion.h"
#include "sms_app.h"

#include "ParserXML.h"

using namespace std;

InterfazEdicion::InterfazEdicion(Mapa* mapa) {
    this->mapa = mapa;
    mapa->bloquearElementos();
}

InterfazEdicion::~InterfazEdicion() {
    delete mapa;
}

void InterfazEdicion::comenzar(ChatDoble* chat) {
    Ventana miVentana = Ventana(ANCHO_VENTANA, ALTO_VENTANA, chat);
    agregarBotoneraInferior(&miVentana);
    string rutaFondo = "./escenario.jpg";
    InterfazMapa * interfazMapa = new InterfazMapa(rutaFondo, rutaFondo, mapa);

    interfazMapa->setAltura(mapa->getAlto());
    interfazMapa->setAncho(mapa->getAncho());
    interfazMapa->setRutinaMouseDown(mapa_mouseDown);
    interfazMapa->setRutinaMouseOver(mapa_mouseOver);
    interfazMapa->setRutinaMouseUp(mapa_mouseUp);
    miVentana.agregarPanel(interfazMapa, Coordenada((long)0, (long)0));

    Mapa* mapaux = new Mapa();

    InterfazMapa * interfazMapaContrincante = new InterfazMapa("escenario.jpg", "escenario.jpg",
        mapaux, .3);

    interfazMapaContrincante->setAltura(0);
    interfazMapaContrincante->setAncho(0);
    miVentana.agregarPanel(interfazMapaContrincante, Coordenada((long)1100, (long)-600));

    miVentana.ejecutar(interfazMapaContrincante, mapa);
}

void InterfazEdicion::agregarBotoneraInferior(Ventana * miVentana) {

    /* Boton Masa */
    Boton * botonMasa = new Boton(MASA, mapa);
    botonMasa->setRutinaMouseDown(botonMasa_mouseDown);
    botonMasa->setHabilitar(mapa->elementoHabilitado(MASA));
    miVentana->agregarPanel(botonMasa, 0);

    /* Boton Rueda */
    Boton * botonRueda = new Boton(RUEDA, mapa);
    botonRueda->setRutinaMouseDown(botonRueda_mouseDown);
    botonRueda->setHabilitar(mapa->elementoHabilitado(RUEDA));
    miVentana->agregarPanel(botonRueda, 0);

    /* Boton PuntoFijo */
    Boton * botonPuntoFijo = new Boton(PUNTO_FIJO, mapa);
    botonPuntoFijo->setRutinaMouseDown(botonPuntoFijo_mouseDown);
    botonPuntoFijo->setHabilitar(mapa->elementoHabilitado(PUNTO_FIJO));
    miVentana->agregarPanel(botonPuntoFijo, 0);

    /* Boton Barra */
}
```

```
Boton * botonBarra = new Boton(BARRA, mapa);
botonBarra->setRutinaMouseDown(botonBarra_mouseDown);
botonBarra->setHabilitar(mapa->elementoHabilitado(BARRA));
miVentana->agregarPanel(botonBarra, 0);

/* Boton PLataforma */
Boton * botonPLataforma = new Boton(PLATAFORMA, mapa);
botonPLataforma->setRutinaMouseDown(botonPlataforma_mouseDown);
botonPLataforma->setHabilitar(mapa->elementoHabilitado(PLATAFORMA));
miVentana->agregarPanel(botonPLataforma, 0);

/* Boton Soga */
Boton * botonSoga = new Boton(SOGA, mapa);
botonSoga->setRutinaMouseDown(botonSoga_mouseDown);
botonSoga->setHabilitar(mapa->elementoHabilitado(SOGA));
miVentana->agregarPanel(botonSoga, 0);

/* Boton Cinta */
Boton * botonCinta = new Boton(CINTA, mapa);
botonCinta->setRutinaMouseDown(botonCinta_mouseDown);
botonCinta->setHabilitar(mapa->elementoHabilitado(CINTA));
miVentana->agregarPanel(botonCinta, 0);

/* Boton Cohete */
Boton * botonCohete = new Boton(COHETE, mapa);
botonCohete->setRutinaMouseDown(botonCohete_mouseDown);
botonCohete->setHabilitar(mapa->elementoHabilitado(COHETE));
miVentana->agregarPanel(botonCohete, 0);

/* Boton Borrar Elemento */
Boton * botonBorrarElemento = new Boton(BORRAR_ELEMENTO, mapa);
botonBorrarElemento->setRutinaMouseDown(botonBorrarElemento_mouseDown);
botonBorrarElemento->setAltura(32);
botonBorrarElemento->setAncho(32);
miVentana->agregarPanel(botonBorrarElemento, 0);

/* Boton Borrar Todo */
Boton * botonBorrarTodo = new Boton(BORRAR_TODO, mapa);
botonBorrarTodo->setRutinaMouseDown(botonBorrarTodo_mouseDown);
botonBorrarTodo->setAltura(32);
botonBorrarTodo->setAncho(32);
miVentana->agregarPanel(botonBorrarTodo, 0);

/* Boton Void */
Boton * botonVoid = new Boton(VOID, mapa);
botonVoid->setRutinaMouseDown(NULL);
botonVoid->setAltura(32);
botonVoid->setAncho(32);
miVentana->agregarPanel(botonVoid, 0);

/* Boton PLAY */
Boton * botonPlay = new Boton(PLAY, mapa);
botonPlay->setRutinaMouseDown(botonPlay_mouseDown);
botonPlay->setAltura(32);
botonPlay->setAncho(32);
miVentana->agregarPanel(botonPlay, 0);

/* Boton SMS */
Boton * botonSms = new Boton(SMS, mapa);
botonSms->setRutinaMouseDown(botonSms_mouseDown);
botonSms->setAltura(32);
botonSms->setAncho(32);
miVentana->agregarPanel(botonSms, 0);

/* Boton Salir */
Boton * botonSalir = new Boton(SALIR, mapa);
botonSalir->setRutinaMouseDown(botonSalir_mouseDown);
botonSalir->setAltura(32);
botonSalir->setAncho(32);
miVentana->agregarPanel(botonSalir, 0);
```

```
/* EVENTOS MAPA */

void mapa_mouseDown(Coordenada punto, InterfazMapa * interfaz) {
    Mapa * mapa = interfaz->getMapa();
    Coordenada puntoCorregido;
    const Elemento * elemento;

    if((mapa->entornoDePuntoDeEnlace(punto, &puntoCorregido) == true)) {
        if(mapa->getNroElementoParaCrear() > 2) { // si es longitudinal lo que se va crear
            mapa->agregarElemento(puntoCorregido);
            if(elemento = mapa->getElementoSeleccionado() == NULL) return;
            if(elemento->es(LONGITUDINAL))
                mapa->setTipoSeleccion(ELEMENTO_SELECCIONADO_POR_EXTREM02);
            return;
        }
    }

    int resul = mapa->seleccionarElemento(punto);
    elemento = mapa->getElementoSeleccionado();
    if(elemento != NULL && elemento->estaBloqueado()) {
        mapa->setTipoSeleccion(ELEMENTO_NO_SELECCIONADO);
        return;
    }
    switch (resul) {

        case ELEMENTO_NO_SELECCIONADO:
            mapa->agregarElemento(punto);
            elemento = mapa->getElementoSeleccionado();
            mapa->setMostrarPuntosDeEnlaceElemento(false);
            if(elemento == NULL) {
                interfaz->vectorDiferencia = punto - interfaz->getPosicion();
                return;
            }
            else if(elemento->es(LONGITUDINAL))
                mapa->setTipoSeleccion(ELEMENTO_SELECCIONADO_POR_EXTREM02);
            else if(elemento->es(CIRCULAR)) {
                mapa->setTipoSeleccion(ELEMENTO_SELECCIONADO_POR_CENTRO);
                interfaz->vectorDiferencia = punto - elemento->getPosicion();
            }
            break;

        case ELEMENTO_SELECCIONADO_POR_CENTRO:
            if(mapa->elementoABorrar == true)
                mapa->eliminarElemento();
            else {
                mapa->setMostrarPuntosDeEnlaceElemento(false);
                interfaz->vectorDiferencia = punto - elemento->getPosicion();
            }
            break;
        case ELEMENTO_SELECCIONADO_POR_EXTREM02:
            mapa->setMostrarPuntosDeEnlaceElemento(false);
            break;
    }
}

void mapa_mouseOver(Coordenada click, InterfazMapa * interfaz) {
    Mapa * mapa = interfaz->getMapa();

    if(!interfaz->is_clicked() || !mapa->hayElementoSeleccionado()) return;
    // A partir de aca el mapa se encuentra clickeado con un elemento seleccionado
    Coordenada clickCorregido((long)0,(long)0);
    int tipoSeleccion = mapa->getTipoSeleccion();
```

```
bool corregirClick = mapa->entornoDePuntoDeEnlace(click, &clickCorregido);

Coordenada punto = (corregirClick == true)? clickCorregido : click;

switch (tipoSeleccion) {

    case ELEMENTO_SELECCIONADO POR CENTRO:
        mapa->moverElementoA(punto - interfaz->vectorDiferencia);
        break;

    case ELEMENTO_SELECCIONADO POR EXTREMO2: // solo los longitudinales caen aqui
        if(mapa->getElementoSeleccionado()->es(COHETE))
            mapa->actualizarAngulo(punto);
        else
            mapa->actualizarModuloAngulo(punto);
        break;

}

void mapa_mouseUp(Coordenada click, InterfazMapa * interfaz) {
    Mapa * mapa = interfaz->getMapa();
    if(!mapa->hayElementoSeleccionado()) return;
    mapa->actualizarEnlaces();
    mapa->setMostrarPuntosDeEnlaceElemento(true);
}

/* EVENTOS BOTON MASA */

void botonMasa_mouseDown(Coordenada cord, Boton * boton) {

    Mapa * miMapa = boton->getMapa();
    miMapa->elementoABorrar = false;

    if(miMapa->getNroElementoParaCrear() != MASA) {
        miMapa->setNroElementoParaCrear(MASA);
    }else{
        miMapa->setNroElementoParaCrear(NINGUN_ELEMENTO);
    }
}

/* EVENTOS BOTON RUEDA */

void botonRueda_mouseDown(Coordenada cord, Boton * boton) {

    Mapa * miMapa = boton->getMapa();
    miMapa->elementoABorrar = false;

    if(miMapa->getNroElementoParaCrear() != RUEDA) {
        miMapa->setNroElementoParaCrear(RUEDA);
    }else{
        miMapa->setNroElementoParaCrear(NINGUN_ELEMENTO);
    }
}

/* EVENTOS BOTON PUNTOFIJO */

void botonPuntoFijo_mouseDown(Coordenada cord, Boton * boton) {

    Mapa * miMapa = boton->getMapa();
    miMapa->elementoABorrar = false;

    if(miMapa->getNroElementoParaCrear() != PUNTO_FIJO) {
        miMapa->setNroElementoParaCrear(PUNTO_FIJO);
    }else{
        miMapa->setNroElementoParaCrear(NINGUN_ELEMENTO);
    }
}
```

```
/* EVENTOS BOTON BARRA */

void botonBarra_mouseDown(Coordenada cord, Boton * boton) {

    Mapa * miMapa = boton->getMapa();
    miMapa->elementoABorrar = false;

    if(miMapa->getNroElementoParaCrear() != BARRA) {
        miMapa->setNroElementoParaCrear(BARRA);
    }else{
        miMapa->setNroElementoParaCrear(NINGUN_ELEMENTO);
    }
}

/* EVENTOS BOTON PLATAFORMA */

void botonPlataforma_mouseDown(Coordenada cord, Boton * boton) {

    Mapa * miMapa = boton->getMapa();
    miMapa->elementoABorrar = false;

    if(miMapa->getNroElementoParaCrear() != PLATAFORMA) {
        miMapa->setNroElementoParaCrear(PLATAFORMA);
    }else{
        miMapa->setNroElementoParaCrear(NINGUN_ELEMENTO);
    }
}

/* EVENTOS BOTON SOGA */

void botonSoga_mouseDown(Coordenada cord, Boton * boton) {

    Mapa * miMapa = boton->getMapa();
    miMapa->elementoABorrar = false;

    if(miMapa->getNroElementoParaCrear() != SOGA) {
        miMapa->setNroElementoParaCrear(SOGA);
    }else{
        miMapa->setNroElementoParaCrear(NINGUN_ELEMENTO);
    }
}

/* EVENTOS BOTON CINTA */

void botonCinta_mouseDown(Coordenada cord, Boton * boton) {

    Mapa * miMapa = boton->getMapa();
    miMapa->elementoABorrar = false;

    if(miMapa->getNroElementoParaCrear() != CINTA) {
        miMapa->setNroElementoParaCrear(CINTA);
    }else{
        miMapa->setNroElementoParaCrear(NINGUN_ELEMENTO);
    }
}

/* EVENTOS BOTON ZONA_LLEGADA */

void botonZonaLlegada_mouseDown(Coordenada cord, Boton * boton) {

    Mapa * miMapa = boton->getMapa();
    miMapa->elementoABorrar = false;

    if(miMapa->getNroElementoParaCrear() != ZONA_LLEGADA) {
        miMapa->setNroElementoParaCrear(ZONA_LLEGADA);
    }else{
        miMapa->setNroElementoParaCrear(NINGUN_ELEMENTO);
    }
}
```

```
}

/* EVENTOS BOTON COHETE */

void botonCohete_mouseDown(Coordenada cord, Boton * boton) {

    Mapa * miMapa = boton->getMapa();

    miMapa->elementoABorrar = false;

    if(miMapa->getNroElementoParaCrear() != COHETE) {
        miMapa->setNroElementoParaCrear(COHETE);
    } else{
        miMapa->setNroElementoParaCrear(NINGUN_ELEMENTO);
    }
}

/* EVENTOS BOTON BORRAR_TODO */

void botonBorrarTodo_mouseDown(Coordenada cord, Boton * boton) {

    Mapa * miMapa = boton->getMapa();
    miMapa->elementoABorrar = false;
    miMapa->setNroElementoParaCrear(NINGUN_ELEMENTO);
    miMapa->eliminarTodosLosElementos();

}

/* EVENTOS BOTON BORRAR_ELEMENTO */

void botonBorrarElemento_mouseDown(Coordenada cord, Boton * boton) {

    Mapa * miMapa = boton->getMapa();

    if(miMapa->elementoABorrar == false)
        miMapa->elementoABorrar = true;
    else
        miMapa->elementoABorrar = false;
    miMapa->setNroElementoParaCrear(NINGUN_ELEMENTO);

}

/* EVENTOS BOTON PLAY */

void botonPlay_mouseDown(Coordenada cord, Boton * boton) {
    Mapa * mapa = boton->getMapa();

    mapa->setTiempoResolucion(boton->pedirTiempoResolucion());
    boton->enviarSenialSimulacion();

}

/* EVENTOS BOTON SALIR */

void botonSalir_mouseDown(Coordenada cord, Boton * boton) {
    boton->enviarSenialSalida();
}

/* EVENTOS BOTON SMS */

void botonSms_mouseDown(Coordenada cord, Boton * boton) {
    boton->enviarSenialSms();

}
```

```
#include "InterfazMapa.h"
#include <iostream>
#include <list>
#include "Elemento.h"
#include "Barra.h"
#include "Masa.h"
#include "Rueda.h"
#include <list>
#include "PuntoDeEnlace.h"

using namespace std;

InterfazMapa::InterfazMapa(const string & rutaImgDefecto,
    const string & rutaImgClickDown, Mapa * mapa) :
    PanelContenedor("Mapa", rutaImgDefecto, rutaImgDefecto, rutaImgDefecto) {
    this->mapa = mapa;
    pf_rutinaMouseDown = NULL;
    pf_rutinaMouseOver = NULL;
    pf_rutinaMouseUp = NULL;

    loadTexture("./ima/mapa/ball.bmp", TEXTURA_MASA);
    loadTexture("./ima/mapa/punto_fijo.bmp", TEXTURA_PUNTO_FIJO);
    loadTexture("./ima/mapa/lona.bmp", TEXTURA_CINTA);
    loadTexture("./ima/mapa/rope.bmp", TEXTURA_SOGA);
    loadTexture("./ima/mapa/metal.bmp", TEXTURA_BARRA);
    loadTexture("./ima/mapa/rocket.bmp", TEXTURA_COHETE);
    loadTexture("./ima/mapa/platform.bmp", TEXTURA_PLATAFORMA);
    loadTexture("./ima/mapa/rocket.bmp", TEXTURA_RUEDA);
    loadTexture("./ima/mapa/llegada.bmp", TEXTURA_ZONA_LLEGADA);

}

InterfazMapa::InterfazMapa(const string & rutaImgDefecto,
    const string & rutaImgClickDown, Mapa * mapa, double zoom) :
    PanelContenedor("Mapa", rutaImgDefecto, rutaImgDefecto, rutaImgDefecto, zoom) {
    this->mapa = mapa;
    pf_rutinaMouseDown = NULL;
    pf_rutinaMouseOver = NULL;
    pf_rutinaMouseUp = NULL;

    loadTexture("./ima/mapa/ball.bmp", TEXTURA_MASA);
    loadTexture("./ima/mapa/punto_fijo.bmp", TEXTURA_PUNTO_FIJO);
    loadTexture("./ima/mapa/lona.bmp", TEXTURA_CINTA);
    loadTexture("./ima/mapa/rope.bmp", TEXTURA_SOGA);
    loadTexture("./ima/mapa/metal.bmp", TEXTURA_BARRA);
    loadTexture("./ima/mapa/rocket.bmp", TEXTURA_COHETE);
    loadTexture("./ima/mapa/platform.bmp", TEXTURA_PLATAFORMA);
    loadTexture("./ima/mapa/rocket.bmp", TEXTURA_RUEDA);
    loadTexture("./ima/mapa/llegada.bmp", TEXTURA_ZONA_LLEGADA);

}

InterfazMapa::~InterfazMapa() {

}

void InterfazMapa::evento_MouseDown(Coordenada punto) {
    PanelContenedor::evento_MouseDown(punto);
    if (pf_rutinaMouseDown != NULL)
        pf_rutinaMouseDown(punto, this);
}

void InterfazMapa::evento_MouseUp(Coordenada punto) {
    PanelContenedor::evento_MouseUp(punto);
    if (pf_rutinaMouseUp != NULL)
        pf_rutinaMouseUp(punto, this);
}
```

```
void InterfazMapa::evento_MouseOver(Coordenada punto) {
    PanelContenedor::evento_MouseOver(punto);
    if (pf_rutinaMouseOver != NULL)
        pf_rutinaMouseOver(punto, this);
}

void InterfazMapa::setRutinaMouseDown(void (*rutina)(Coordenada, InterfazMapa *)) {
    pf_rutinaMouseDown = rutina;
}

void InterfazMapa::setRutinaMouseUp(void (*rutina)(Coordenada, InterfazMapa *)) {
    pf_rutinaMouseUp = rutina;
}

void InterfazMapa::setRutinaMouseOver(void (*rutina)(Coordenada, InterfazMapa *)) {
    pf_rutinaMouseOver = rutina;
}

Mapa * InterfazMapa::getMapa() const {
    return mapa;
}

void InterfazMapa::setMapa(Mapa* mapa)
{
    if (this->mapa != NULL)
        delete this->mapa;

    this->mapa = mapa;
}

void InterfazMapa::imprimir() {
    PanelContenedor::imprimir();

    list<Elemento*> lista = this->mapa->getListaElementos();

    list<Elemento*>:: iterator it;

    for (it = lista.begin(); it != lista.end(); it++) {
        if ((*it)->es(CIRCULAR))
            imprimirElementoCircular( (ElementoCircular*) (*it) );

        else if ((*it)->es(LONGITUDINAL))
            imprimirElementoLongitudinal( (ElementoLongitudinal*) (*it) );
        else if ((*it)->es(ZONA_LLEGADA))
            imprimirZonaLlegada( (Llegada*) (*it) );
    }
}

void InterfazMapa::imprimirElementoCircular(ElementoCircular * circular) {
    ulong radio = circular->getMagnitud();
    int textura= TEXTURA_MASA;

    if (circular->es(MASA))
        textura = TEXTURA_MASA;
    if (circular->es(PUNTO_FIJO)) {
        textura = TEXTURA_PUNTO_FIJO;
        radio = 4;
    }
    if (circular->es(PUNTO_ENLACE)) {
        textura = TEXTURA_PUNTO_FIJO;
        radio = 3;
    }
    if (circular->es(RUEDA))
        textura = TEXTURA_RUEDA;

    glBindTexture(GL_TEXTURE_2D, _texturas[textura]);
    glBegin(GL_TRIANGLE_FAN);
```

```

double x, y;
x = cambioX(circular->getPosicion().x) + getPosicion().x/TX;
y = cambioY(circular->getPosicion().y) - getPosicion().y/TY;

for (int j = 0; j <= POLY_SIDES; j++) {

    glTexCoord2i( 0, 0);
    glVertex3f(x, y, 0.0f);
    glTexCoord2i( 1, 0);
    glVertex3f(x, y, 0.0f);

    glTexCoord2i( 1, 1);
    glVertex3f(x + radio/TX * cos(j * (2 * PI
        / POLY_SIDES)), y + radio/TY* sin(j * (2
        * PI / POLY_SIDES)), 0.0f);

    j++;
    glTexCoord2i( 0, 1);
    glVertex3f(x + radio/TX * cos(j * (2 * PI
        / POLY_SIDES)), y + radio/TY * sin(j * (2
        * PI / POLY_SIDES)), 0.0f);
    j--;
}

glEnd();

if (circular->es(RUEDA) && circular->getMostrarPuntosDeEnlace()) {
    list <PuntoDeEnlace*> _enlaces = ((Rueda*)circular)->getListaEnlaces();
    list <PuntoDeEnlace*>::iterator it;
    for(it = _enlaces.begin(); it != _enlaces.end() ; it++) {
        imprimirElementoCircular(*it);
    }
}

void InterfazMapa::imprimirElementoLongitudinal(ElementoLongitudinal * elem) {
    int punta = 0;
    int textura= TEXTURA_MASA;
    if (elem->es(BARRA))
        textura = TEXTURA_BARRA;
    else if (elem->es(CINTA))
        textura = TEXTURA_CINTA;
    else if (elem->es(SOGA))
        textura = TEXTURA_SOGA;
    else if (elem->es(COHETE)) {
        textura = TEXTURA_COHETE;
        punta = 10;
    }
    else if (elem->es(PLATAFORMA))
        textura = TEXTURA_PLATAFORMA;

    Coordenada extrOpPrima = elem->getExtremoOpuesto() -
    Coordenada((long) (punta * cos(Angulo::aRadianes(elem->getAngulo())))) , -(long)( punta * sin
(Angulo::aRadianes(elem->getAngulo()))));

    double aX = cambioX(elem->getPosicion().x) - (elem->getAncho()/(2.0*TX)) * cos
(Angulo::aRadianes(90-elem->getAngulo()));
    double aY = cambioY(elem->getPosicion().y) + (elem->getAncho()/(2.0*TY)) * sin
(Angulo::aRadianes(90-elem->getAngulo()));

    double bX = cambioX(extrOpPrima.x) -
    (elem->getAncho()/(2.0*TX)) * cos(Angulo::aRadianes(90-elem->getAngulo()));
    double bY = cambioY(extrOpPrima.y) +
    (elem->getAncho()/(2.0*TY)) * sin(Angulo::aRadianes(90-elem->getAngulo()));
}

```

```
double cX = (elem->getAncho()/TX) *
    sin(Angulo::aRadianes(elem->getAngulo())) + aX;
double cY = aY - (elem->getAncho()/TY) *
    cos(Angulo::aRadianes(elem->getAngulo()));

double dX = cambioX(extrOpPrima.x) +
    (elem->getAncho()/(2.0*TX)) * sin(Angulo::aRadianes(elem->getAngulo()) );

double dY = cambioY(extrOpPrima.y) -
    (elem->getAncho()/(2.0*TX)) * cos(Angulo::aRadianes(elem->getAngulo()) );

aX += getPosicion().x/TX;
bX += getPosicion().x/TX;
cX += getPosicion().x/TX;
dX += getPosicion().x/TX;

aY -= getPosicion().y/TY;
bY -= getPosicion().y/TY;
cY -= getPosicion().y/TY;
dY -= getPosicion().y/TY;

glBindTexture(GL_TEXTURE_2D, _texturas[textura]);
glBegin(GL_QUADS);

glTexCoord2i( 0, 0);
glVertex3f(cX, cY, 0.0f);
glTexCoord2i( 1, 0);
glVertex3f(aX, aY, 0.0f);
glTexCoord2i( 1, 1);
glVertex3f(bX, bY, 0.0f);
glTexCoord2i( 0, 1);
glVertex3f(dX, dY, 0.0f);

glEnd();

if (elem->es(COHETE)) {
    glBegin(GL_QUADS);

    glTexCoord2i( 0, 0);
    glVertex3f(dX, dY, 0.0f);
    glTexCoord2i( 1, 0);
    glVertex3f(bX, bY, 0.0f);
    glTexCoord2i( 1, 1);
    glVertex3f(cambioX(elem->getPosicion().x + elem->getMagnitud() * cos(-Angulo::aRadianes(elem->getAngulo()) + getPosicion().x), cambioY(elem->getPosicion().y + elem->getMagnitud() * sin(-Angulo::aRadianes(elem->getAngulo()) + getPosicion().y), 0.0f));
    glTexCoord2i( 0, 1);
    glVertex3f(cambioX(elem->getPosicion().x + elem->getMagnitud() * cos(-Angulo::aRadianes(elem->getAngulo()) + getPosicion().x), cambioY(elem->getPosicion().y + elem->getMagnitud() * sin(-Angulo::aRadianes(elem->getAngulo()) + getPosicion().y), 0.0f));

    glEnd();
}

if(elem->getMostrarPuntosDeEnlace() ) {
    list <PuntoDeEnlace*> _enlaces = elem->getListaEnlaces();
    list <PuntoDeEnlace*>::iterator it;
    for(it = _enlaces.begin(); it != _enlaces.end() ; it++) {
        imprimirElementoCircular(*it);
    }
}

void InterfazMapa::imprimirZonaLlegada(Llegada * llegada) {
    glColor4d(1.0, 1.0, 4.0, 0.4);
    glEnable (GL_BLEND);
    glBlendFunc (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
    glBindTexture( GL_TEXTURE_2D, _texturas[TEXTURA_ZONA_LLEGADA]);
    glBegin(GL_QUADS);
```

```
glTexCoord2i( 0, 0);
glVertex3f(cambioX(llegada->getPosicion().x + getPosicion().x), cambioY(llegada->getPosicion()
().y + getPosicion().y), 0.0f);
glTexCoord2i( 1, 0);
glVertex3f(cambioX(llegada->getPosicion().x + getPosicion().x), cambioY(llegada->getPosicion()
().y + getPosicion().y + (long)llegada->getMagnitud()), 0.0f);
glTexCoord2i( 1, 1);
glVertex3f(cambioX(llegada->getPosicion().x + getPosicion().x + (long)llegada->getMagnitud()),
cambioY(llegada->getPosicion().y + getPosicion().y + (long)llegada->getMagnitud()), 0.0f);
glTexCoord2i( 0, 1);
glVertex3f(cambioX(llegada->getPosicion().x + getPosicion().x + (long)llegada->getMagnitud()),
cambioY(llegada->getPosicion().y + getPosicion().y), 0.0f);

glEnd();
glColor4d(1.0, 1.0, 1.0, 1.0);

}
```

```
#include "Llegada.h"

Llegada::Llegada(Coordenada posicion) : Elemento ("ZonaLlegada", posicion, 80, 0) {

}

int Llegada::seleccionar(Coordenada punto) const {
    if((punto.x > getPosicion().x) && (punto.x < (getPosicion().x + (long)getMagnitud())))
        if((punto.y > getPosicion().y) && (punto.y < (getPosicion().y + (long)getMagnitud())))
            return ELEMENTO_SELECCIONADO_POR_CENTRO;
    return ELEMENTO_NO_SELECCIONADO;
}
```

```
#include "PanelContenedor.h"
#include "loader.h"
#include "Angulo.h"
#include "constantes.h"
#include "constantes_sms.h"

#include <ctime>
#include <list>
#define ROPES 5
#define LONAS 6
#define RUEDAS 2
#define COHETES 7
#define METAL 3
#define PLATAFORMAS 4
#define PTO_FIJO 1

using namespace std;

unsigned calculaComp(double largo,double &sep){
    double min = 0.03;
    unsigned i = 0;
    while (true){
        if ( (i*min<largo) && ((i+1)*min>=largo) )
            break;
        i++;
    }
    sep = largo / i;
    return i;
}

void drawCintaLona(SpringMassSystem& sms, double x,double y,double largo,double angulo){
    ElementManager* manager;
    manager = sms.getElementManager();

    int ini, fin,fspr;
    double sep = 0.04;
    int longitud = calculaComp(largo,sep);

    fspr = sms.getNumSprings();
    ini = sms.getNumMasses();
    fin = ini + longitud;
    for (int i=0;i<=longitud;i++)
        sms.addCollMass(2.0, Vector2<>(x+i*sep*cos(angulo), y+i*sep*sin(angulo)), 0.01);

    for (int i=ini ;i<fin ;i++)
        sms.addSpring(i,i+1,LONA_K,0);

    manager->addLona(ini,fin);
    sms.addEnganche(ini);
    sms.addEnganche(fin);

    ini = (manager->getLonas()).size()-1;
}

void drawMetalBar(SpringMassSystem& sms, double x,double y,double largo,double angulo){
    ElementManager* manager;
    manager = sms.getElementManager();
    double sep = 0.04;
    int longitud = calculaComp(largo,sep);

    int ini,fin,first,last;
    ini = sms.getNumMasses();
    first = ini;
    fin = ini + longitud;

    for (int i=0;i<=longitud;i++)
        sms.addMass(2.0, Vector2<>(x+i*sep*cos(angulo), y+i*sep*sin(angulo)));

    last = sms.getNumMasses()-1;
    for (int i=ini ;i<fin ;i++)
        sms.addSpring(i,i+1,METAL_K,0);
```

```
x += METAL_WIDTH * sin(angulo);
y -= METAL_WIDTH * cos(angulo) ;

ini = sms.getNumMasses();
fin = ini + longitud;

for (int i=0;i<=longitud;i++)
    sms.addMass(2.0, Vector2<>(x+i*sep*cos(angulo), y+i*sep*sin(angulo)));

for (int i=ini ;i<fin ;i++)
    sms.addSpring(i,i+1,METAL_K,0);

for (int i=ini ;i<=fin ;i++)
    sms.addSpring(i,i-longitud-1,METAL_K,0);
for (int i=ini+1 ;i<=fin ;i++)
    sms.addSpring(i,i-longitud-2,METAL_K,0);

manager->addMetalBar(first,sms.getNumMasses()-1);
sms.addEnganche(first);
sms.addEnganche(last);
}

void drawPlataforma(SpringMassSystem& sms, double x,double y,double largo,double angulo){
ElementManager* manager;
manager = sms.getElementManager();
double sep = 0.04;
int longitud = calculaComp(largo,sep);

int ini,fin,first,last;
ini = sms.getNumMasses();
first = ini;
fin = ini + longitud;

for (int i=0;i<=longitud;i++)
    sms.addCollMass(2.0, Vector2<>(x+i*sep*cos(angulo), y+i*sep*sin(angulo)), 0.01);

last = sms.getNumMasses()-1;
for (int i=ini ;i<fin ;i++)
    sms.addSpring(i,i+1,METAL_K,0);

x += METAL_WIDTH * sin(angulo);
y -= METAL_WIDTH * cos(angulo) ;

ini = sms.getNumMasses();
fin = ini + longitud;

for (int i=0;i<=longitud;i++)
    sms.addMass(2.0, Vector2<>(x+i*sep*cos(angulo), y+i*sep*sin(angulo)));

for (int i=ini ;i<fin ;i++)
    sms.addSpring(i,i+1,METAL_K,0);

for (int i=ini ;i<=fin ;i++)
    sms.addSpring(i,i-longitud-1,METAL_K,0);
for (int i=ini+1 ;i<=fin ;i++)
    sms.addSpring(i,i-longitud-2,METAL_K,0);

manager->addPlataforma(first,sms.getNumMasses()-1);
sms.addEnganche(first);
sms.addEnganche(last);
}

void drawRueda(SpringMassSystem& sms, double x,double y,double radio){
ElementManager* manager;
manager = sms.getElementManager();

unsigned sides = 12 ;
int ini = sms.getNumMasses();
sms.addCollMass( 0,Vector2<>(x,y),0.05);
double dx,dy;
```

```
for (unsigned j = 0; j < sides; j++){
    dx = radio * cos(j * (2.0 * PI / sides));
    dy = radio * sin(j * (2.0 * PI / sides));
    sms.addCollMass( 1.5,
        Vector2<>(x + dx, y + dy),
        0.01);
}
for (unsigned j = ini+1; j < ini + sides; j++){
    sms.addSpring(j,j+1,RUEDA_K,0);
    sms.addSpring(ini,j,RUEDA_K,0);
}
sms.addSpring(ini+1,ini + sides,RUEDA_K,0);
sms.addSpring(ini,ini + sides,RUEDA_K,0);
manager->addRueda(ini,sms.getNumMasses()-1);

for (size_t i = ini + 1; i<=(ini + sides); i++)
    sms.addEnganche(i);

}

void drawCohete(SpringMassSystem& sms, double x,double y,double largo,double angulo){
    ElementManager* manager;
    double xprime,yprime;
    x-=ROCKET_WIDTH*0.5*sin(angulo);
    y+=ROCKET_WIDTH*0.5*cos(angulo);

    xprime = x;
    yprime = y;

    manager = sms.getElementManager();
    int ini, fin,first;
    ini = sms.getNumMasses();
    first = ini;
    double sep = 0.04;
    int longitud = calculaComp(largo,sep);

    fin = ini + longitud - 1;
    for (int i=0;i<longitud;i++)
        sms.addCollMass(1, Vector2<>(x+i*sep*cos(angulo), y+i*sep*sin(angulo)), 0.01);

    for (int i=ini ;i<fin ;i++)
        sms.addSpring(i,i+1,ROCKET_K,0);

    ini = sms.getNumMasses();
    fin = ini + longitud - 1;

    xprime = x + ROCKET_WIDTH * sin(angulo);
    yprime = y - ROCKET_WIDTH * cos(angulo);

    for (int i=0;i<longitud;i++)
        sms.addCollMass(1, Vector2<>(xprime+i*sep*cos(angulo), yprime+i*sep*sin(angulo)), 0.01);

    for (int i=ini ;i<fin ;i++)
        sms.addSpring(i,i+1,ROCKET_K,0);

    for (int i=ini ;i<=fin ;i++)
        sms.addSpring(i,i-longitud,ROCKET_K,0);

    for (int i=ini ;i<fin ;i++)
        sms.addSpring(i,i-longitud+1,ROCKET_K,0);

    xprime = x+ROCKET_WIDTH*0.5*sin(angulo);
    yprime = y-ROCKET_WIDTH*0.5*cos(angulo);

    sms.addCollMass(1, Vector2<>(xprime+longitud*sep*cos(angulo), yprime+longitud*sep*sin(angulo)), 0.03);
    ini = sms.getNumMasses();
    sms.addSpring(ini-1,first+longitud-1,ROCKET_K,0);
    sms.addSpring(ini-1,first+2*longitud-1,ROCKET_K,0);
```

```
    sms.addCollMass(1, Vector2<>(xprime, yprime), 0.01);
    sms.addSpring(ini,first,ROCKET_K,0);
    sms.addSpring(ini,first+longitud,ROCKET_K,0);
    manager->addCohete(first,sms.getNumMasses()-1);
    sms.addEnganche(sms.getNumMasses()-1);
    sms.addEnganche(sms.getNumMasses()-2);
}

void drawRope(SpringMassSystem& sms, double x,double y,double largo,double angulo){
    ElementManager* manager;
    manager = sms.getElementManager();
    int ini, fin;
    double sep = 0.04;
    int longitud = calculaComp(largo,sep);

    ini = sms.getNumMasses();
    fin = ini + longitud;

    for (int i=0;i<=longitud;i++)
        sms.addMass(3, Vector2<>(x+i*sep*cos(angulo), y+i*sep*sin(angulo)));

    for (int i=ini ;i<fin ;i++)
        sms.addSpring(i,i+1,ROPE_K,0);

    manager->addRope(ini,fin);
    sms.addEnganche(ini);
    sms.addEnganche(fin);
}

void drawFixedSpot(SpringMassSystem& sms, double x,double y){
    ElementManager* manager;
    manager = sms.getElementManager();
    sms.addMass(0,Vector2<>(x, y));
    manager->addPuntoFijo(sms.getNumMasses()-1,sms.getNumMasses()-1);
    sms.addEnganche(sms.getNumMasses()-1);
}

void drawCrashableFixedSpot(SpringMassSystem& sms, double x,double y){
    sms.addCollMass(0,Vector2<>(x, y),0.03);
}

void drawBall(SpringMassSystem& sms, double x,double y,double radio){
    ElementManager* manager;
    manager = sms.getElementManager();
    unsigned sides = 16 ;
    int ini = sms.getNumMasses();
    sms.addCollMass( 0.5,Vector2<>(x,y),0.05);
    double dx,dy;
    for (unsigned j = 0; j < sides; j++){
        dx = radio * cos(j * (2.0 * PI / sides));
        dy = radio * sin(j * (2.0 * PI / sides));
        sms.addCollMass( 1.5,
                        Vector2<>(x + dx, y + dy),
                        0.009);
    }
    for (unsigned j = ini+1; j < ini + sides; j++){
        sms.addSpring(j,j+1,BALL_K,0);
        sms.addSpring(ini,j,BALL_K,0);
    }

    sms.addSpring(ini+1,ini + sides,BALL_K,0);
    sms.addSpring(ini,ini + sides,BALL_K,0);
    manager->addMainBall(ini,sms.getNumMasses()-1);
}

void drawZonaLlegada(SpringMassSystem& sms,double x ,double y,double width,double heigth ){
    ElementManager* manager;
    manager = sms.getElementManager();

    manager->addZonaLlegada(x,y,width,heigth);
```

```
}

void fixMass(SpringMassSystem& sms, int pos){
    sms.fixMass(pos);
}

    double cambioX(long x) {
        return (x - WIDTH/2.0)/tX;
}

    double cambioY(long y) {
        return (-y + HEIGHT/2.0)/tY;
}
void instanciar(SpringMassSystem& sms)
{
    const double TX = 300.0;
    Mapa* mapa = sms.getMapa();

    list <Elemento*> elementos = mapa->getListaElementos();
    list <Elemento*>::iterator it;

    for(it = elementos.begin(); it != elementos.end(); it++) {

        if((*it)->es(MASA))
            drawBall(sms, cambioX((*it)->getPosicion().x) ,cambioY((*it)->getPosicion().y),
(*it)->getMagnitud()/TX);
        if((*it)->es(RUEDA))
            drawRueda(sms, cambioX((*it)->getPosicion().x) ,cambioY((*it)->getPosicion
().y),(*it)->getMagnitud()/TX);
        if((*it)->es(SOGA))
            drawRope(sms, cambioX((*it)->getPosicion().x) ,cambioY((*it)->getPosicion().y),
(*it)->getMagnitud()/TX,Angulo::aRadianes((*it)->getAngulo()));
        if((*it)->es(CINTA))
            drawCintaLona(sms, cambioX((*it)->getPosicion().x) ,cambioY((*it)->getPosicion
().y),(*it)->getMagnitud()/TX,Angulo::aRadianes((*it)->getAngulo()));
        if((*it)->es(PLATAFORMA))
            drawPlataforma(sms, cambioX((*it)->getPosicion().x) ,cambioY((*it)->getPosicion
().y),(*it)->getMagnitud()/TX,Angulo::aRadianes((*it)->getAngulo()));
        if((*it)->es(BARRA))
            drawMetalBar(sms, cambioX((*it)->getPosicion().x) ,cambioY((*it)->getPosicion
().y),(*it)->getMagnitud()/TX,Angulo::aRadianes((*it)->getAngulo()));
        if((*it)->es(COHETE))
            drawCohete(sms, cambioX((*it)->getPosicion().x) ,cambioY((*it)->getPosicion
().y),(*it)->getMagnitud()/TX,Angulo::aRadianes((*it)->getAngulo()));
        if((*it)->es(PUNTO_FIJO))
            drawFixedSpot(sms, cambioX((*it)->getPosicion().x) ,cambioY((*it)->getPosicion
().y));
        if((*it)->es(ZONA_LLEGADA))
            drawZonaLlegada(sms,cambioX((*it)->getPosicion().x) ,cambioY((*it)->getPosicion
().y),(*it)->getMagnitud()/TX,(*it)->getMagnitud()/TX );
    }

    sms.setRocketTime(mapa->getTiempoCohete());
    //cout << mapa->getTiempoCohete()<< endl;
    sms.checkRoutine();
}

void (*instance[]) (SpringMassSystem&) = {instanciar};

void load(SpringMassSystem& sms)
{
    srand(static_cast<unsigned>(time(0)));
    int n = sizeof(instance) / sizeof(instance[0]);
    instance[rand() % n](sms);
}
```

```
#include "Mapa.h"
#include "ElementoEnlazable.h"
#include "ElementoLongitudinal.h"
#include "PuntoDeEnlace.h"
#include <math.h>
#include "Angulo.h"
using namespace std;

Mapa::Mapa() {
    this->ancho = 1024;
    this->alto = 550;
    elemento = NULL;
    nroElementoParaCrear = NINGUN_ELEMENTO;
    tipoSeleccion = ELEMENTO_NO_SELECCIONADO;
    elementoABorrar = false;

    for(int i = 0; i < CANTIDAD_ELEMENTOS; i++) {
        elementosHabilitados[i] = true; // todos habilitados por defecto
        precio[i] = 0; // son gratis por default
    }

    plata = 100;
    puntos = -1;
}

Mapa::Mapa(ulong ancho, ulong alto) {
    this->ancho = ancho;
    this->alto = alto;
    elemento = NULL;
    nroElementoParaCrear = NINGUN_ELEMENTO;
    tipoSeleccion = ELEMENTO_NO_SELECCIONADO;
    elementoABorrar = false;

    for(int i = 0; i < CANTIDAD_ELEMENTOS; i++) {
        elementosHabilitados[i] = true; // todos habilitados por defecto
        precio[i] = 0; // son gratis por default
    }

    plata = 100;
    puntos = -1;
}

Mapa::~Mapa() {
    desbloquearElementos();
    eliminarTodosLosElementos();
}

ulong Mapa::getAncho() const {
    return ancho;
}

ulong Mapa::getAlto() const {
    return alto;
}

void Mapa::setNroElementoParaCrear(int elemento) {
    nroElementoParaCrear = elemento;
    if(elemento== NINGUN_ELEMENTO) elemento = NULL;
}

int Mapa::getNroElementoParaCrear() const {
    return nroElementoParaCrear;
}

list <Elemento*> Mapa::getListaElementos() const {
    return _elementos;
}

bool Mapa::agregarElemento(Coordenada posicion) {
    FabricaDeElementos fabrica;
    Elemento * aux;
```

```
aux = fabrica.crear(nroElementoParaCrear, posicion);
if (aux !=NULL) {
    _elementos.push_back(aux);
    if (nroElementoParaCrear != ZONA_LLEGADA)
        plata -= precio[nroElementoParaCrear];
    elemento = aux;
    return true;
} else
    return false;
}

size_t Mapa::getCantidadElementos() {
    return _elementos.size();
}

int Mapa::seleccionarElemento(Coordenada punto) {
    list<Elemento*>::iterator it;

    int resultado= ELEMENTO_NO_SELECCIONADO;

    for (it = _elementos.begin(); it != _elementos.end(); it++)
        if ((*it)->seleccionar(punto)) != ELEMENTO_NO_SELECCIONADO) {
            elemento = (*it);
            break;
        }

    if(resultado == ELEMENTO_NO_SELECCIONADO) {
        elemento = NULL;
    }

    tipoSeleccion = resultado;
    return resultado;
}

bool Mapa::hayElementoSeleccionado() const {
    return ((elemento != NULL)? true : false);
}

int Mapa::getTipoSeleccion() const {
    return tipoSeleccion;
}

void Mapa::setTipoSeleccion(const int tipo) {
    tipoSeleccion = tipo;
}

const Elemento * Mapa::getElementoSeleccionado() const {
    return elemento;
}

bool Mapa::entornoDePuntoDeEnlace(Coordenada punto, Coordenada * puntoExacto) {

    list <ElementoEnlazable*> _enlaces = getTodosLosEnlaces();
    list <ElementoEnlazable*>::iterator it;

    for (it = _enlaces.begin(); it != _enlaces.end(); it++) {
        if((*it)->seleccionar(punto) != ELEMENTO_NO_SELECCIONADO) {
            *puntoExacto = (*it)->getPosicion();
            return true;
        }
    }

    return false;
}

void Mapa::setExtremoOpuesto(const Coordenada punto) {
    if(elemento != NULL) {
        if(elemento->es(LONGITUDINAL))
            ((ElementoLongitudinal*)elemento)->setExtremoOpuesto(punto);
    }
}
```

```
Coordenada Mapa::getExtremoOpuesto() const {
    if(elemento != NULL) {
        return ((ElementoLongitudinal*)elemento)->getExtremoOpuesto();
    }
    return Coordenada( (long) -1, (long)-1 );
}

bool Mapa::moverElementoA(const Coordenada posicion) {
    if(elemento != NULL)
        elemento->setPosicion(posicion);
    return true;
}

bool Mapa::rotarElementoA(long angulo) {
    Angulo::corregir(&angulo);
    if(elemento != NULL)
        elemento->setAngulo(angulo);
    return true;
}

bool Mapa::agrandarElementoA(long longitud) {
    if(elemento != NULL)
        elemento->setMagnitud(longitud);
    return true;
}

Coordenada Mapa::actualizarModuloAngulo(Cordenada click) {
    if(elemento != NULL) {
        if( (click - elemento->getPosicion()).modulo < LONGITUD_MINIMA)
            actualizarAngulo(click);
        else {
            if(elemento->es(BARRA) || elemento->es(PLATAFORMA)) {
                if( ( (click - elemento->getPosicion()).modulo < LONGITUD_MAXIMA))
                    ((ElementoLongitudinal *)elemento)->setExtremoOpuesto(click);
                else
                    actualizarAngulo2(click);
            } else
                ((ElementoLongitudinal *)elemento)->setExtremoOpuesto(click);
        }
    }
    return click;
}

Coordenada Mapa::actualizarAngulo(Cordenada click) {
    if(elemento != NULL) {
        int longitudMinima = LONGITUD_MINIMA;

        if(elemento->es(COHETE))
            longitudMinima = LONGITUD_COHETE;
        Coordenada resta = click - elemento->getPosicion();
        if(resta.angulo > 180)
            resta.setAngulo(-resta.angulo);
        resta.setModulo(longitudMinima);
        ((ElementoLongitudinal *)elemento)->setExtremoOpuesto(elemento->getPosicion() + resta);
    }
    return click;
}

Coordenada Mapa::actualizarAngulo2(Cordenada click) {
    if(elemento != NULL) {
        int longitudMinima = LONGITUD_MAXIMA;

        if(elemento->es(COHETE))
            longitudMinima = LONGITUD_COHETE;
        Coordenada resta = click - elemento->getPosicion();
        if(resta.angulo > 180)
```

```
        resta.setAngulo(-resta.angulo);
        resta.setModulo(longitudMinima);
        ((ElementoLongitudinal *)elemento)->setExtremoOpuesto(elemento->getPosicion() + resta);
    }
    return click;
}

void Mapa::eliminarElemento() {
    if(elemento != NULL) {
        if(!elemento->estaBloqueado()) {
            _elementos.remove(elemento);
            plata += precio[elemento->getTipo()];
            delete elemento;
            elemento = NULL;
        }
    }
}

void Mapa::eliminarTodosLosElementos() {
    list<Elemento*>::iterator it;

    for (it = _elementos.begin(); it != _elementos.end(); ) {
        elemento = *it;
        if(!elemento->estaBloqueado()) {
            eliminarElemento();
            it = _elementos.begin();
        }
        else
            it++;
    }

    nroElementoParaCrear = NINGUN_ELEMENTO;
    elemento = NULL;
}

void Mapa::actualizarEnlaces() {
    list<Elemento*>::iterator it;
    for (it = _elementos.begin(); it != _elementos.end(); it++) {
        if((*it)->es(LONGITUDINAL)) {
            ElementoLongitudinal * ele = (ElementoLongitudinal *) *it;
            ele->actualizarPuntosDeEnlace(1);
        }
        else if( (*it)->es(RUEDA)) {
            ((Rueda*)(*it))->actualizarPuntosDeEnlace();
        }
    }
}

list <ElementoEnlazable*> Mapa::getTodosLosEnlaces() {
    list <ElementoEnlazable*> lista;

    list<Elemento*>::iterator it;

    for (it = _elementos.begin(); it != _elementos.end(); it++) {
        if( (*it)->es(ENLAZABLE) )
            lista.push_back((ElementoEnlazable*) *it);
        else if( (*it)->es(LONGITUDINAL) )
            ElementoLongitudinal * eleLong = (ElementoLongitudinal *) *it;
            list <PuntoDeEnlace*> _puntosDeEnlace = eleLong->getListaEnlaces();
            list <PuntoDeEnlace*>::iterator it2;
            for(it2 = _puntosDeEnlace.begin(); it2 != _puntosDeEnlace.end(); it2++)
                lista.push_back((ElementoEnlazable*) *it2);
        }
        else if( (*it)->es(RUEDA) ) {
            list <PuntoDeEnlace*> _puntosDeEnlace = ((Rueda*)(*it))->getListaEnlaces();
            list <PuntoDeEnlace*>::iterator it2;
            for(it2 = _puntosDeEnlace.begin(); it2 != _puntosDeEnlace.end(); it2++)
                lista.push_back((ElementoEnlazable*) *it2);
        }
    }
}
```

```
        }
    }

    return lista;
}

void Mapa::imprimirTodosLosEnlaces() {
    list <ElementoEnlazable*> lista = getTodosLosEnlaces();

    list<ElementoEnlazable*>::iterator it;

    for (it = lista.begin(); it != lista.end(); it++) {
        (*it)->getPosicion().imprimir();
    }
}

void Mapa::imprimir() {
    cout << "PLATA: " << plata << endl;
    list<Elemento*>::iterator it;
    for (it = _elementos.begin(); it != _elementos.end(); it++) {
        (*it)->imprimir();
    }
}

void Mapa::setMostrarPuntosDeEnlaceElemento(const bool b) {
    if(elemento != NULL) {
        elemento->setMostrarPuntosDeEnlace(b);
    }
}

void Mapa::bloquearElementos() {
    list<Elemento*>::iterator it;
    for (it = _elementos.begin(); it != _elementos.end(); it++) {
        (*it)->bloquear();
    }
}

void Mapa::desbloquearElementos() {
    list<Elemento*>::iterator it;
    for (it = _elementos.begin(); it != _elementos.end(); it++) {
        (*it)->desbloquear();
    }
}

void Mapa::deshabilitarElemento(const int nroElemento) {
    elementosHabilitados[nroElemento] = false;
}

void Mapa::habilitarElemento(const int nroElemento) {
    elementosHabilitados[nroElemento] = true;
}

bool Mapa::elementoHabilitado(const int nroElemento) const {
    return elementosHabilitados[nroElemento];
}

void Mapa::setNombre(const string & nombre) {
    this->nombre = nombre;
}

string Mapa::getNombre() const {
    return nombre;
}

void Mapa::setPlata(const int plata) {
    this->plata = plata;
}
```

```
int Mapa::getPlata() const {
    return plata;
}

void Mapa::setPuntos(const int puntos) {
    this->puntos = puntos;
}

int Mapa::getPuntos() const {
    return puntos;
}

void Mapa::setPrecio(int nroElemento, int precio) {
    this->precio[nroElemento] = precio;
}

int Mapa::getPrecio(int nroElemento) const{
    return this->precio[nroElemento];
}

int Mapa::getTiempoCohete() const {
    return tiempoCohete;
}
void Mapa::setTiempoCohete(int tiempo) {
    this->tiempoCohete = tiempo;
}

void Mapa::setTiempoResolucion(const long tiempo) {
    this->tiempoResolucion = tiempo;
}

long Mapa::getTiempoResolucion() const {
    return tiempoResolucion;
}

int Mapa::getSumaDePrecios() const {
    int suma = 0;
    for(int i = 0; i < CANTIDAD_ELEMENTOS; i++) {
        suma += precio[i];
    }
    return suma;
}
```

```
#include "Masa.h"
using namespace std;

Masa::Masa() :
    ElementoCircular("Circular:Masa", Coordenada(long(100), long(100)), 10, 0) {}

Masa::Masa(Cordenada posicion, ulong magnitud) :
    ElementoCircular("Circular:Masa", posicion, magnitud, 0) {}

Masa::~Masa() {
```

```
#include "PanelContenedor.h"
#include <iostream>
#include "constantes.h"
using namespace std;
PanelContenedor::PanelContenedor(const string & nombre,
                                  const string & rutaImgDefecto, const string & rutaImgClickDown,
                                  const string & rutaImgMouseOver) {

    this->nombre = nombre;

    ancho = getAncho(rutaImgDefecto);
    alto = getAlto(rutaImgDefecto);

    loadTexture(rutaImgDefecto.c_str(), TEXTURA_DEFECTO);
    loadTexture(rutaImgClickDown.c_str(), TEXTURA_MOUSEDOWN);
    loadTexture(rutaImgMouseOver.c_str(), TEXTURA_MOUSEOVER);
    loadTexture(rutaImgMouseOver.c_str(), TEXTURA_INHABILITADO);
    texturaActual = TEXTURA_DEFECTO;

    this->id = 0;

    posicion.setX(0);
    posicion.setY(0);
    pantalla = NULL;
    visible = true;
    habilitado = true;
    clicked = false;
    mouseOver = false;
    TX = 300.0;
    TY = 300.0;
}

PanelContenedor::PanelContenedor(const string & nombre,
                                  const string & rutaImgDefecto, const string & rutaImgClickDown,
                                  const string & rutaImgMouseOver, double zoom) {

    this->nombre = nombre;

    ancho = getAncho(rutaImgDefecto);
    alto = getAlto(rutaImgDefecto);

    loadTexture(rutaImgDefecto.c_str(), TEXTURA_DEFECTO);
    loadTexture(rutaImgClickDown.c_str(), TEXTURA_MOUSEDOWN);
    loadTexture(rutaImgMouseOver.c_str(), TEXTURA_MOUSEOVER);
    loadTexture(rutaImgMouseOver.c_str(), TEXTURA_INHABILITADO);
    texturaActual = TEXTURA_DEFECTO;

    this->id = 0;

    posicion.setX(0);
    posicion.setY(0);
    pantalla = NULL;
    visible = true;
    habilitado = true;
    clicked = false;
    mouseOver = false;
    TX = 300.0 / zoom;
    TY = 300.0 / zoom;
}

ulong PanelContenedor::getAncho(const string & rutaImg) {
    SDL_Surface * surface = IMG_Load(rutaImg.c_str());
    return (ulong) surface->w;
}

ulong PanelContenedor::getAlto(const string & rutaImg) {
    SDL_Surface * surface = IMG_Load(rutaImg.c_str());
    return (ulong) surface->h;
}

void PanelContenedor::loadTexture(const char * path, int tipoTextura) {
    GLuint textura;
```

```
GLint nOfColors;
GLenum texture_format = 0;

SDL_Surface * surface = IMG_Load(path);

if (surface == NULL) {
    cout << "No se pudo abrir la imagen: " << path << endl;
    return;
}

if ( (surface->w & (surface->w - 1)) != 0) {
    printf("ERROR en imagen, el ancho no es potencia de 2\n");
}

if ( (surface->h & (surface->h - 1)) != 0) {
    printf("ERROR en imagen, el alto no es potencia de 2\n");
}

nOfColors = surface->format->BytesPerPixel;
if (nOfColors == 4)
{
    if (surface->format->Rmask == 0x000000ff)
        texture_format = GL_RGBA;
    else
        texture_format = GL_BGRA;
} else if (nOfColors == 3)
{
    if (surface->format->Rmask == 0x000000ff)
        texture_format = GL_RGB;
    else
        texture_format = GL_BGR;
} else {
    printf("warning: the image is not truecolor.. this will probably break\n");
}

glGenTextures( 1, &textura);

glBindTexture( GL_TEXTURE_2D, textura);

glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

glTexImage2D( GL_TEXTURE_2D, 0, nOfColors, surface->w, surface->h, 0,
texture_format, GL_UNSIGNED_BYTE, surface->pixels);
_texturas.insert(pair<int, GLuint>(tipoTextura, textura));

}

PanelContenedor::~PanelContenedor() {

}

double PanelContenedor::cambioX(long x) {
    return (x - pantalla->w/2.0)/TX;
}

double PanelContenedor::cambioY(long y) {
    return (-y + pantalla->h/2.0)/TY;
}

void PanelContenedor::imprimirImagenDeFondo() {

    glBindTexture( GL_TEXTURE_2D, _texturas[texturaActual]);
    glBegin(GL_QUADS);

    glTexCoord2i( 0, 0);
    glVertex3f(cambioX(posicion.x), cambioY(posicion.y), 0.0f);
    glTexCoord2i( 1, 0);
    glVertex3f(cambioX(posicion.x), cambioY(posicion.y + alto), 0.0f);
    glTexCoord2i( 1, 1);
    glVertex3f(cambioX(posicion.x + ancho), cambioY(posicion.y + alto), 0.0f);
    glTexCoord2i( 0, 1);
    glVertex3f(cambioX(posicion.x + ancho), cambioY(posicion.y), 0.0f);
}
```

```
    glEnd();  
}  
  
void PanelContenedor::imprimir() {  
    imprimirImagenDeFondo();  
    list <PanelContenedor*>::iterator it;  
  
    for (it = _paneles.begin(); it != _paneles.end(); it++) {  
        (*it)->imprimir();  
    }  
}  
  
void PanelContenedor::setAltura(const ulong alto) {  
    this->alto = alto;  
}  
  
void PanelContenedor::setAncho(const ulong ancho) {  
    this->ancho = ancho;  
}  
  
ulong PanelContenedor::getAltura() const {  
    return alto;  
}  
  
ulong PanelContenedor::getAncho() const {  
    return ancho;  
}  
  
void PanelContenedor::setPosicion(const Coordenada posicion) {  
    this->posicion = posicion;  
}  
  
Coordenada PanelContenedor::getPosicion() const {  
    return posicion;  
}  
  
void PanelContenedor::setPantalla(SDL_Surface * pantalla) {  
    this->pantalla = pantalla;  
}  
  
void PanelContenedor::setVentana(Ventana * ventana) {  
    this->ventana = ventana;  
}  
  
string PanelContenedor::getNombre() const {  
    return nombre;  
}  
  
int PanelContenedor::getId() const {  
    return id;  
}  
  
void PanelContenedor::setId(const int id) {  
    this->id = id;  
}  
  
void PanelContenedor::setMouseOver(const bool b) {  
    this->mouseOver = b;  
}  
  
void PanelContenedor::agregarPanel(PanelContenedor * panel,  
                                    Coordenada posicionEnPanel) {  
    panel->setPosicion(this->posicion + posicionEnPanel);  
    panel->setPantalla(pantalla);  
    _paneles.push_back(panel);  
}  
  
void PanelContenedor::agregarPanel(PanelContenedor * panel, int lugar) {  
    Coordenada pos;  
    _paneles.push_back(panel);
```

```
switch (lugar) {
    case 1:
        pos.setX(this->ancho / _paneles.size());
        pos.setY(this->alto + ((this->alto - panel->getAltura()) / 2));
        break;
}

panel->setPantalla(pantalla);
panel->setPosicion(pos);

}

void PanelContenedor::setImagenDeFondo(int imagen) {
    texturaActual = imagen;
}

bool PanelContenedor::pertenece(Coordenada punto) {

    int bordeIzquierdo = posicion.x;
    int bordeDerecho = posicion.x + ancho;
    int bordeSuperior = posicion.y;
    int bordeInferior = posicion.y + alto;

    if ( (punto.x >= bordeIzquierdo) && (punto.x < bordeDerecho) && (punto.y
                                > bordeSuperior) && (punto.y < bordeInferior))
        return true;
    return false;
}

bool PanelContenedor::seEjecutaEventoEn(const Coordenada punto, int evento) {

    if (!pertenece(punto)) {
        return false;
    }

    list<PanelContenedor*>::iterator it;
    for (it = _paneles.begin(); it != _paneles.end(); it++)
        if ((*it)->pertenece(punto))
            if ((*it)->seEjecutaEventoEn(punto, evento))
                return false;
}

if(habilitado) {
    switch (evento) {
        case SDL_MOUSEBUTTONDOWN:
            evento_MouseDown(punto - getPosicion());
            break;
        case SDL_MOUSEMOTION:
            evento_MouseOver(punto - getPosicion());
            break;
        case SDL_MOUSEBUTTONUP:
            evento_MouseUp(punto - getPosicion());
            break;
    }
    return true;
}
return false;
}

void PanelContenedor::evento_MouseDown(Coordenada punto) {
    if (clicked==false) {
        clicked = true;
        if (texturaActual != TEXTURA_MOUSEDOWN)
            texturaActual = TEXTURA_MOUSEDOWN;
        else
            texturaActual = TEXTURA_DEFECTO;
    }
}

void PanelContenedor::evento_MouseOver(Coordenada punto) {

    if (mouseOver == false) {
```

```
        mouseOver = true;
        texturaActual = TEXTURA_MOUSEOVER;
    }

void PanelContenedor::evento_MouseUp(Coordenada punto) {
    if (clicked == true) {
        clicked = false;
    }
}

bool PanelContenedor::is_clicked() const {
    return clicked;
}

bool PanelContenedor::is_over() const {
    return mouseOver;
}

bool PanelContenedor::estaHabilitado() const {
    return habilitado;
}

void PanelContenedor::setHabilitar(const bool b) {
    habilitado = b;
}
```

```
#include "ParserXML.h"

#include <libxml/parser.h>
#include <libxml/tree.h>
#include <iostream>
#include <list>
#include <fstream>
using namespace std;

Mapa* ParserXML::obtenerMapaArchivo (const std::string& rutaEntrada) const
{
    xmlTextReaderPtr reader;
    reader = xmlReaderFromFile(rutaEntrada.c_str(), NULL, 0);
    return obtenerMapa(reader);
}

Mapa* ParserXML::obtenerMapa (xmlTextReaderPtr reader) const
{
    Mapa* mapa = new Mapa();

    int ret;
    if (reader != NULL)
    {
        ret = xmlTextReaderRead(reader);
        while (ret == 1)
        {
            analizarNodo(mapa, reader);
            ret = xmlTextReaderRead(reader);
        }
    }
    mapa->actualizarEnlaces();
    mapa->setNroElementoParaCrear(NINGUN_ELEMENTO);
    return mapa;
}

int ParserXML::nroElemento(const std::string& tipo) const
{
    string nom = tipo.substr(tipo.find_last_of(":")+1,tipo.length()-tipo.find_last_of(":")-1);

    if (nom.compare("Masa") == 0)
        return MASA;

    if (nom.compare("Rueda") == 0)
        return RUEDA;

    if (nom.compare("PuntoFijo") == 0)
        return PUNTO_FIJO;

    if (nom.compare("Barra") == 0)
        return BARRA;

    if (nom.compare("Plataforma") == 0)
        return PLATAFORMA;

    if (nom.compare("Soga") == 0)
        return SOGA;

    if (nom.compare("Cinta") == 0)
        return CINTA;

    if (nom.compare("Cohete") == 0)
        return COHETE;

    if (nom.compare("ZonaLlegada") == 0)
        return ZONA_LLEGADA;

    return NINGUN_ELEMENTO;
}
```

```
void ParserXML::procesarElemento(Mapa* mapa, xmlTextReaderPtr reader) const
{
    const xmlChar *value;
    string tipo;
    int nroTipo;
    char* tipoAux;
    long angulo;
    ulong magnitud;
    Coordenada coor;

    xmlTextReaderRead(reader);

    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader); //leo el tipo
    value = xmlTextReaderConstValue(reader);
        tipoAux = (char*) value;
        tipo = tipoAux;
        nroTipo = nroElemento(tipo);
    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader);

    mapa->setNroElementoParaCrear(nroTipo);

    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader); //leo la magnitud
    value = xmlTextReaderConstValue(reader);
        magnitud = atoi ((const char*) value);
    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader);

    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader); //leo el angulo
    value = xmlTextReaderConstValue(reader);
        angulo = atoi ((const char*) value);
    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader);

    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader); //leo la posicion

    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader); //leo x
    value = xmlTextReaderConstValue(reader);
        coor.x = atoi ((const char*) value);
    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader);

    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader); //leo y
    value = xmlTextReaderConstValue(reader);
        coor.y = atoi ((const char*) value);
    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader);

    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader);

    mapa->agregarElemento(coor);

    if ((nroTipo >= BARRA) && (nroTipo <= COHETE))
    {
        Coordenada extremoOpuesto;

        xmlTextReaderRead(reader);
        xmlTextReaderRead(reader);

        xmlTextReaderRead(reader);
        xmlTextReaderRead(reader);
        //leo x
        value = xmlTextReaderConstValue(reader);
```

```
extremoOpuesto.x = atoi((const char*) value);
xmlTextReaderRead(reader);
xmlTextReaderRead(reader);

xmlTextReaderRead(reader);
xmlTextReaderRead(reader);
//leo y
value = xmlTextReaderConstValue(reader);
extremoOpuesto.y = atoi((const char*) value);
xmlTextReaderRead(reader);
xmlTextReaderRead(reader);

xmlTextReaderRead(reader);
xmlTextReaderRead(reader);

mapa->setExtremoOpuesto(extremoOpuesto);

}

mapa->agrandarElementoA(magnitud);
mapa->rotarElementoA(angulo);

// xmlTextReaderRead(reader);
// xmlTextReaderRead(reader);

xmlTextReaderRead(reader);

}

void ParserXML::procesarHabilitados(Mapa* mapa, xmlTextReaderPtr reader) const
{
    const xmlChar *value;
    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader);
    value = xmlTextReaderConstValue(reader); //leo masa
    if (strcmp((const char*)value,"NO") == 0)
        mapa->deshabilitarElemento(MASA);
    else
        mapa->habilitarElemento(MASA);
    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader);

    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader);
    value = xmlTextReaderConstValue(reader); //leo punto fijo
    if (strcmp((const char*)value,"NO") == 0)
        mapa->deshabilitarElemento(PUNTO_FIJO);
    else
        mapa->habilitarElemento(PUNTO_FIJO);
    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader);

    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader);
    value = xmlTextReaderConstValue(reader); //leo rueda
    if (strcmp((const char*)value,"NO") == 0)
        mapa->deshabilitarElemento(RUEDA);
    else
        mapa->habilitarElemento(RUEDA);
    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader);

    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader);
    value = xmlTextReaderConstValue(reader); //leo barra
    if (strcmp((const char*)value,"NO") == 0)
        mapa->deshabilitarElemento(BARRA);
```

```
else
    mapa->habilitarElemento(BARRA);
xmlTextReaderRead(reader);
xmlTextReaderRead(reader);

xmlTextReaderRead(reader);
xmlTextReaderRead(reader);
value = xmlTextReaderConstValue(reader); //leo plataforma
if (strcmp((const char*)value,"NO") == 0)
    mapa->deshabilitarElemento(PLATAFORMA);
else
    mapa->habilitarElemento(PLATAFORMA);
xmlTextReaderRead(reader);
xmlTextReaderRead(reader);

xmlTextReaderRead(reader);
xmlTextReaderRead(reader);
value = xmlTextReaderConstValue(reader); //leo soga
if (strcmp((const char*)value,"NO") == 0)
    mapa->deshabilitarElemento(SOGA);
else
    mapa->habilitarElemento(SOGA);
xmlTextReaderRead(reader);
xmlTextReaderRead(reader);

xmlTextReaderRead(reader);
xmlTextReaderRead(reader);
value = xmlTextReaderConstValue(reader); //leo cinta
if (strcmp((const char*)value,"NO") == 0)
    mapa->deshabilitarElemento(CINTA);
else
    mapa->habilitarElemento(CINTA);
xmlTextReaderRead(reader);
xmlTextReaderRead(reader);

xmlTextReaderRead(reader);
xmlTextReaderRead(reader);
value = xmlTextReaderConstValue(reader); //leo cohete
if (strcmp((const char*)value,"NO") == 0)
    mapa->deshabilitarElemento(COHETE);
else
    mapa->habilitarElemento(COHETE);
xmlTextReaderRead(reader);
xmlTextReaderRead(reader);

xmlTextReaderRead(reader);

}

void ParserXML::procesarCostos(Mapa* mapa, xmlTextReaderPtr reader) const
{
    const xmlChar *value;
    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader);
    value = xmlTextReaderConstValue(reader); //leo masa
    mapa->setPrecio(MASA,atoi((const char*)value));
    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader);

    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader);
    value = xmlTextReaderConstValue(reader); //leo punto fijo
    mapa->setPrecio(PUNTO_FIJO,atoi((const char*)value));

    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader);

    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader);
}
```

```
    value = xmlTextReaderConstValue(reader); //leo rueda
    mapa->setPrecio(RUEDA,atoi((const char*)value));
    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader);

    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader);
    value = xmlTextReaderConstValue(reader); //leo barra

    mapa->setPrecio(BARRA,atoi((const char*)value));
    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader);

    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader);
    value = xmlTextReaderConstValue(reader); //leo plataforma

    mapa->setPrecio(PLATAFORMA,atoi((const char*)value));
    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader);

    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader);
    value = xmlTextReaderConstValue(reader); //leo soga

    mapa->setPrecio(SOGA,atoi((const char*)value));
    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader);

    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader);
    value = xmlTextReaderConstValue(reader); //leo cinta

    mapa->setPrecio(CINTA,atoi((const char*)value));
    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader);

    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader);
    value = xmlTextReaderConstValue(reader); //leo cohete

    mapa->setPrecio(COHETE,atoi((const char*)value));
    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader);

    xmlTextReaderRead(reader);
}

}
```

```
void ParserXML::analizarNodo(Mapa* mapa, xmlTextReaderPtr reader) const
{
    const xmlChar *name, *value;
    char* valueAux;

    name = xmlTextReaderConstName(reader);

    if (strcmp((const char*)name,"nombre") == 0)
    {
        xmlTextReaderRead(reader);
        value = xmlTextReaderConstValue(reader);
        valueAux = (char*) value;
        mapa->setNombre(valueAux);//asignar el nombre al mapa
        xmlTextReaderRead(reader);
        xmlTextReaderRead(reader);
    }
    else if (strcmp((const char*)name,"plata") == 0)
    {
        xmlTextReaderRead(reader);
    }
}
```

```
        value = xmlTextReaderConstValue(reader);
        valueAux = (char*) value;
        mapa->setPlata(atoi(valueAux));
        xmlTextReaderRead(reader);
    }
    else if (strcmp((const char*)name, "elemento") == 0)
        procesarElemento(mapa, reader);

    else if (strcmp((const char*)name, "habilitados") == 0)
        procesarHabilitados(mapa, reader);

    else if (strcmp((const char*)name, "costes") == 0)
        procesarCostos(mapa, reader);

    else if (strcmp((const char*)name, "TiempoCohete") == 0)
    {
        xmlTextReaderRead(reader);
        value = xmlTextReaderConstValue(reader);
        valueAux = (char*) value;
        mapa->setTiempoCohete(atoi(valueAux));
        xmlTextReaderRead(reader);
    }
}

Mapa* ParserXML::obtenerMapaMemoria (const std::string& xml) const
{
    xmlTextReaderPtr reader;

    reader = xmlReaderForMemory(xml.c_str(), xml.length(), NULL, NULL, 0);

    return obtenerMapa(reader);
}

void ParserXML::obtenerXMLArchivo (const std::string& rutaSalida, const Mapa* mapa) const
{
    string xml;
    obtenerXMLMemoria(xml, mapa);
    ofstream arch(rutaSalida.c_str());
    arch.write(xml.c_str(),xml.length());
    arch.close();
}

void ParserXML::enteroAString(std::string& cadena, ulong entero) const
{
    if (entero == 0)
        cadena = "0";
    else
        cadena = "";
    while (entero != 0)
    {
        switch (entero %10)
        {
            case 0: cadena="0"+cadena; break;
            case 1: cadena="1"+cadena; break;
            case 2: cadena="2"+cadena; break;
            case 3: cadena="3"+cadena; break;
            case 4: cadena="4"+cadena; break;
            case 5: cadena="5"+cadena; break;
            case 6: cadena="6"+cadena; break;
            case 7: cadena="7"+cadena; break;
            case 8: cadena="8"+cadena; break;
            case 9: cadena="9"+cadena; break;
        }
        entero /= 10;
    }
}

void ParserXML::imprimirElementos(std::string& xml, const Mapa* mapa) const
```

```
{  
    list<Elemento*> lista = mapa->getListaElementos();  
    Elemento* actual;  
    string nro;  
    Coordenada coorActual;  
    list<Elemento*>::iterator it = lista.begin();  
    while (it != lista.end())  
    {  
        actual = *it;  
        xml += " ";  
        xml += "<elemento>\n";  
  
        xml += " " <tip>;  
        xml += actual->getNombre();  
        xml += "</tip>\n";  
  
        xml += " " <magnitud>;  
        enteroAString(nro,actual->getMagnitud());  
        xml += nro;  
        xml += "</magnitud>\n";  
  
        xml += " " <angulo>;  
        if ((actual->getAngulo() < 0)  
            enteroAString(nro,actual->getAngulo()+360);  
        else  
            enteroAString(nro,actual->getAngulo());  
        xml += nro;  
        xml += "</angulo>\n";  
  
        coorActual = actual->getPosicion();  
  
        xml += " " <posicion>\n";  
  
        xml += " " <x>;  
        enteroAString(nro,coorActual.x);  
        xml += nro;  
        xml += "</x>\n";  
  
        xml += " " <y>;  
        enteroAString(nro,coorActual.y);  
        xml += nro;  
        xml += "</y>\n";  
  
        xml += " " </posicion>\n";  
  
        if (actual->es(LONGITUDINAL))  
        {  
            coorActual = actual->getExtremoOpuesto();  
  
            xml += " " <extremo_opuesto>\n";  
  
            xml += " " <x>;  
            enteroAString(nro,coorActual.x);  
            xml += nro;  
            xml += "</x>\n";  
  
            xml += " " <y>;  
            enteroAString(nro,coorActual.y);  
            xml += nro;  
            xml += "</y>\n";  
  
            xml += " " </extremo_opuesto>\n";  
        }  
  
        xml += " ";  
        xml += "</elemento>\n";  
    }  
}
```

```
        it++;
    }

}

void ParserXML::obtenerXMLMemoria (const Mapa* mapa) const
{
    string plata;
    enteroAString(plata,mapa->getPlata());

    xml.clear();

/*
    xml += "<?xml version=";
    xml += "'";
    xml += "1.0";
    xml += "'";
    xml += " encoding=";
    xml += "'";
    xml += "UTF-8";
    xml += "'";
    xml += "?>\n"; */

    xml += "<mapa>\n";
    xml += "      ";
    xml += "<nombre>";
    xml += mapa->getNombre();
    xml += "</nombre>\n";
    xml += "      ";
    xml += "<plata>";
    xml += plata;
    xml += "</plata>\n";
    xml += "      ";
    xml += "<elementos>\n";
    imprimirElementos(xml,mapa);
    xml += "</elementos>\n";
    imprimirHabilitados(xml,mapa);
    imprimirCostes(xml,mapa);
    imprimirTiempo(xml,mapa);
    xml += "      ";
    xml += "</mapa>";
}

void ParserXML::imprimirHabilitados(const Mapa* mapa) const
{
    xml += "      <habilitados>\n";

    xml += "          <Masa>";
    if (mapa->elementoHabilitado(MASA))
        xml += "SI";
    else
        xml += "NO";
    xml += "</Masa>\n";

    xml += "          <Punto_Fijo>";
    if (mapa->elementoHabilitado(PUNTO_FIJO))
        xml += "SI";
    else
        xml += "NO";
    xml += "</Punto_Fijo>\n";

    xml += "          <Rueda>";
    if (mapa->elementoHabilitado(RUEDA))
        xml += "SI";
    else
        xml += "NO";
    xml += "</Rueda>\n";

    xml += "          <Barra>";
    if (mapa->elementoHabilitado(BARRA))
        xml += "SI";
    else
```

```
        xml += "NO";
xml += "</Barra>\n";

xml += "                <Plataforma>";
if (mapa->elementoHabilitado(PLATAFORMA))
    xml += "SI";
else
    xml += "NO";
xml += "</Plataforma>\n";

xml += "                <Soga>";
if (mapa->elementoHabilitado(SOGA))
    xml += "SI";
else
    xml += "NO";
xml += "</Soga>\n";

xml += "                <Cinta>";
if (mapa->elementoHabilitado(CINTA))
    xml += "SI";
else
    xml += "NO";
xml += "</Cinta>\n";

xml += "                <Cohete>";
if (mapa->elementoHabilitado(COHETE))
    xml += "SI";
else
    xml += "NO";
xml += "</Cohete>\n";

xml += "            </habilitados>\n";
}

void ParserXML::imprimirCostes(std::string& xml, const Mapa* mapa) const
{
    string nro;

    xml += "            <costes>\n";

    xml += "                <Masa>";
    enteroAString(nro, mapa->getPrecio(MASA));
    xml += nro;
    xml += "</Masa>\n";

    xml += "                <Punto_Fijo>";
    enteroAString(nro, mapa->getPrecio(PUNTO_FIJO));
    xml += nro;
    xml += "</Punto_Fijo>\n";

    xml += "                <Rueda>";
    enteroAString(nro, mapa->getPrecio(RUEDA));
    xml += nro;
    xml += "</Rueda>\n";

    xml += "                <Barra>";
    enteroAString(nro, mapa->getPrecio(BARRA));
    xml += nro;
    xml += "</Barra>\n";

    xml += "                <Plataforma>";
    enteroAString(nro, mapa->getPrecio(PLATAFORMA));
    xml += nro;
    xml += "</Plataforma>\n";

    xml += "                <Soga>";
    enteroAString(nro, mapa->getPrecio(SOGA));
    xml += nro;
    xml += "</Soga>\n";

    xml += "                <Cinta>";
```

```
    enteroAString(nro, mapa->getPrecio(CINTA));
    xml += nro;
    xml += "</Cinta>\n";

    xml += "          <Cohete>";
    enteroAString(nro, mapa->getPrecio(COHETE));
    xml += nro;
    xml += "</Cohete>\n";

    xml += "      </costes>\n";
}

void ParserXML::imprimirTiempo(std::string& xml, const Mapa* mapa) const
{
    string nro;
    enteroAString(nro, mapa->getTiempoCohete());
    xml += "          <TiempoCohete>";
    xml += nro;
    xml += "</TiempoCohete>\n";
}
```

```
#include "Plataforma.h"

Plataforma::Plataforma() :
    ElementoLongitudinal("Longitudinal:Plataforma", Coordenada(long(10), long(10)), 25, 0, 10) {}

Plataforma::Plataforma(Cordenada posicion, ulong magnitud, long angulo, long ancho) :
    ElementoLongitudinal("Longitudinal:Plataforma", posicion, magnitud, angulo, ancho) {}

Plataforma::~Plataforma() {}
```

```
#include "PuntoDeEnlace.h"

PuntoDeEnlace::PuntoDeEnlace() :
    ElementoEnlazable("Circular:Enlazable:PuntoDeEnlace", Coordenada(long(100), long(100)), 15, 0,
NULL) {}

PuntoDeEnlace::PuntoDeEnlace(Coordenada posicion, Elemento * poseedor) :
    ElementoEnlazable("Circular:Enlazable:PuntoDeEnlace", posicion, 10, 0, poseedor) {}

PuntoDeEnlace::~PuntoDeEnlace() {}
```

```
#include "PuntoFijo.h"

PuntoFijo::PuntoFijo() :
    ElementoEnlazable("Circular:Enlazable:PuntoFijo", Coordenada(long(100), long(100)), 10, 0,
NULL) {}

PuntoFijo::PuntoFijo(Coordenada posicion, ulong magnitud) :
    ElementoEnlazable("Circular:Enlazable:PuntoFijo", posicion, magnitud, 0, NULL) {}

PuntoFijo::~PuntoFijo() {}
```

```
#include "Reloj.h"
#include <iostream>
using namespace std;

#define TAM_LETRA_X 0.05
#define TAM_LETRA_Y 0.1

void Reloj::cargarTexturas()
{
    string path;

    char i;
/*    for (i = 'a' ; i <= 'z' ; i++)
    {
        path = "letras/letra__";
        path += i;
        path += ".bmp";
        loadTexture(path.c_str(),i);
    }*/
    for (i = '0' ; i <= '9' ; i++)
    {
        path = "letras/letra__";
        path += i;
        path += ".bmp";
        loadTexture(path.c_str(),i);
    }
    loadTexture("letras/letra_s.bmp",'s');
    loadTexture("letras/vacio.bmp",' ');
    loadTexture("letras/letra_dos_puntos.bmp",';');
    loadTexture("letras/letra_punto.bmp",'.');
    loadTexture("letras/letra_guion.bmp",'-' );
    loadTexture("letras/letra_pesos.bmp",'$');

}

void Reloj::imprimirReloj(const std::string& reloj, float x, float y)
{
    unsigned int i;

    for ( i = 0 ; i < reloj.length() ; i++)
    {
        imprimirCaracter(reloj[i],x,y);
        x += TAM_LETRA_X;
    }

}

void Reloj::imprimirCaracter(char letra, float x, float y)
{

    map<char,GLuint>::iterator it = texturas.find(letra);
    if (it == texturas.end())
        return;

    GLuint textura = it->second;

    glBindTexture( GL_TEXTURE_2D, textura);

    glBegin(GL_QUADS);
    {
        glTexCoord2i( 0, 0 );
        glVertex3f( x, y, 0.0f );
        glTexCoord2i( 0, 1 );
        glVertex3f( x, y - TAM_LETRA_Y, 0.0f );
        glTexCoord2i( 1, 1 );
        glVertex3f( x + TAM_LETRA_X, y - TAM_LETRA_Y, 0.0f );
        glTexCoord2i( 1, 0 );
        glVertex3f( x + TAM_LETRA_X, y, 0.0f );
    }
    glEnd();
}
```

```
void Reloj::loadTexture(const char * path, char letra) {
    GLuint texture;
    GLint nOfColors;
    GLenum texture_format = 0;
    SDL_Surface *surface;

    if ( (surface = SDL_LoadBMP(path)) ) {

        // Check that the image's width is a power of 2
        if ( (surface->w & (surface->w - 1)) != 0 ) {
            printf("ERROR en imagen, el ancho no es potencia de 2\n");
        }

        // Also check if the height is a power of 2
        if ( (surface->h & (surface->h - 1)) != 0 ) {
            printf("ERROR en imagen, el alto no es potencia de 2\n");
        }

        // get the number of channels in the SDL surface
        nOfColors = surface->format->BytesPerPixel;
        if (nOfColors == 4)      // contains an alpha channel
        {
            if (surface->format->Rmask == 0x000000ff)
                texture_format = GL_RGBA;
            else
                texture_format = GL_BGRA;
        } else if (nOfColors == 3)      // no alpha channel
        {
            if (surface->format->Rmask == 0x000000ff)
                texture_format = GL_RGB;
            else
                texture_format = GL_BGR;
        } else {
            printf("warning: the image is not truecolor.. this will probably break\n");
            // this error should not go unhandled
        }
    }

    // Have OpenGL generate a texture object handle for us
    glGenTextures( 1, &texture );

    // Bind the texture object
    glBindTexture( GL_TEXTURE_2D, texture );

    // Set the texture's stretching properties
    glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR );
    glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR );

    // Edit the texture object's image data using the information SDL_Surface gives us
    glTexImage2D( GL_TEXTURE_2D, 0, nOfColors, surface->w, surface->h, 0,
                  texture_format, GL_UNSIGNED_BYTE, surface->pixels );
    texturas.insert(make_pair(letra, texture));
}

else {
    printf("SDL could not load image.bmp: %s\n", SDL_GetError());
    SDL_Quit();
}
```

```
#include "Rueda.h"
#include <math.h>
using namespace std;

Rueda::Rueda() :
    ElementoCircular("Circular:Rueda", Coordenada(long(100), long(100)), 10, 0) {
}

Rueda::Rueda(Coordenada posicion, ulong magnitud) :
    ElementoCircular("Circular:Rueda", posicion, magnitud, 0) {
}

Rueda::~Rueda() {
    eliminarEnlaces();
    _enlaces.clear();
}

void Rueda::actualizarPuntosDeEnlace() {
    eliminarEnlaces();
    _enlaces.clear();

    ulong radio = getMagnitud();
    ulong x = getPosition().x;
    ulong y = getPosition().y;

    for (int j = 0; j <= POLY_SIDES; j += 2) {
        Coordenada pos;
        pos.setX(x + radio * cos(j * (2 * PI
                                         / POLY_SIDES)));
        pos.setY(y + radio * sin(j * (2 * PI
                                         / POLY_SIDES)));

        _enlaces.push_back(new PuntoDeEnlace(pos, this));
    }
}

void Rueda::eliminarEnlaces() {
    list <PuntoDeEnlace*>::iterator it;
    for(it = _enlaces.begin(); it != _enlaces.end(); it++) {
        delete (*it);
    }
    _enlaces.clear();
}

void Rueda::imprimirEnlaces() {
    list <PuntoDeEnlace*>::iterator it;
    cout << "Cantidad de enlaces: " << _enlaces.size() << endl;
    for(it = _enlaces.begin(); it != _enlaces.end(); it++) {
        (*it)->getPosition().imprimir();
    }
    cout << endl << endl;
}

std::list<PuntoDeEnlace*> Rueda::getListaEnlaces() {
    return _enlaces;
}
```

```
#include "sms_app.h"
#include "loader.h"
#include <cmath>
#include <fstream>
#include <SDL.h>
#include <SDL_opengl.h>
#include <sstream>
#include <vector>
#include <iostream>
#include "Angulo.h"
#include "constantes.h"
#include "constantes_sms.h"
#include "hiloCron.h"
#define SHOW_ALL 0
#define SHOW_TEXT 1
#define ANCHO 0.02

using namespace std;
namespace
{
    const double MASS_SIZE = 0.02;
    const int POLY_SIDE = 12;

    const double MAX_DT = 0.006;

    const std::string CONFIG_FILENAME = "config.txt";
}

SMSApp::SMSApp(int argc, Mapa* mapa) :
config_(CONFIG_FILENAME),
sms_(config_),
video_(config_, "./escenario.jpg"),
ticks_(0),
physMult_(config_.read<int>("App.PhysMultiplier"))
{
    cronometro = new HiloCron();
    sms_.setCronometro(cronometro);
    reloj.cargarTexturas();
    tiempoLlegada = NULL;
    estabaEnLlegada = false;
    sms = false;
    sms_.setMapa(mapa);
    if (argc == 0)
        load(sms_);
    else
    {
        sms=true;
        load(sms_);
    }
}

SMSApp::~SMSApp()
{
    delete(cronometro);
}

bool SMSApp::estaEnLlegada(){
    if ((sms_.getElementManager())->getMainBalls().size()){
        MassProxy masa = sms_.getMassProxy((sms_.getElementManager())->getMainBalls().at(0)->getFirst());
        Vector2<> m = masa.getPos();
        double rad = RADIOMASA;
        vector<ZonaLlegada*> llegadas = (sms_.getElementManager())->getZonasLlegada();
        ZonaLlegada* z;

        for (size_t i= 0 ; i<llegadas.size(); i++)
        {
            z = llegadas.at(i);
            if (
                ((m.x - rad) > z->getX()) &&
```

```
((m.x + rad) < (z->getX() + z->getWidth()) ) &&
((m.y + rad) < z->getY()) &&
((m.y - rad) > (z->getY() - z->getHeigth() ) )
)
    return true;
}
}
return false;
}

bool SMSApp::ganaNivel(){

if (estaEnLlegada()){
    if (!estabaEnLlegada){
        tiempoLlegada = new HiloCron();
        tiempoLlegada->start();
        estabaEnLlegada = true;
    }
    else {
        if (tiempoLlegada->getMilisegundos() > TIEMPOGANAR)
            return true;
    }
}
else {
    estabaEnLlegada = false;
    if (tiempoLlegada){
        tiempoLlegada->terminar();
        tiempoLlegada->join();
        delete(tiempoLlegada);
        tiempoLlegada = NULL;
    }
}
return false;
}

double SMSApp::run()
{
    bool gano = false;
    //HiloCron cronometro;
    bool exit_now = false;

    ticks_ = SDL_GetTicks();

    SDL_Event ev;

    cronometro->start();
    while (!exit_now)
    {
        while (SDL_PollEvent(&ev)){
            if (ev.type == SDL_QUIT)
                exit_now = true;
            if (ev.type == SDL_KEYDOWN)
                if ( ev.key.keysym.sym == SDLK_ESCAPE )
                    exit_now = true;
                else if (!strcmp(SDL_GetKeyName(ev.key.keysym.sym), "s"))
                    sms = !sms;
        }
        uint32_t newTicks = SDL_GetTicks();
        double dt;

        dt = MAX_DT;

        for (int i = 0; i < physMult_; i++)
            sms_.rk4Step(dt / physMult_);
        ticks_ = newTicks;

        /*****IMPRIMIR RELOJ*****/
        double tiempo = cronometro->getMilisegundos() / TIME_C;
        stringstream ss;

        ss << fixed;
        ss.precision( 2 );
        ss << tiempo;
    }
}
```

```

*****FIN IMPRIMIR RELOJ*****
string time = ss.str();
time += " s.";
drawFrame(time);
if (gano = ganaNivel())
    exit_now = true;

}
double tiempo = cronometro->getMilisegundos() / TIME_C;
cronometro->terminar();
cronometro->join();

if (gano)
    return tiempo;
return 0;
}

double adjX(long x) {
    return (x - WIDTH/2.0)/tX;
}

double adjY(long y) {
    return (-y + HEIGHT/2.0)/tY;
}
void SMSApp::drawFrame(string &time)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    glBindTexture( GL_TEXTURE_2D, video_.fondo);
    glBegin(GL_QUADS);

    glTexCoord2i( 0, 0);
    glVertex3f(adjX(0), adjY(0), 0.0f);
    glTexCoord2i( 1, 0);
    glVertex3f(adjX(0), adjY(HEIGHT), 0.0f);
    glTexCoord2i( 1, 1);
    glVertex3f(adjX(0 + WIDTH), adjY(HEIGHT), 0.0f);
    glTexCoord2i( 0, 1);
    glVertex3f(adjX(0 + WIDTH), adjY(0), 0.0f);

    glEnd();

    int status;
    status=SHOW_ALL;
    if (!sms)
        status=SHOW_TEXT;

    reloj.imprimirReloj(time, CLOCK_X, CLOCK_Y);

    // draws all springs
    glBegin(GL_LINES);
    if (status==SHOW_ALL)
        for (size_t i = 0; i < sms_.getNumSprings(); i++)
    {
        SpringProxy sp = sms_.getSpringProxy(i);

        Vector2<> startPos = sp.getStartPosition();
        Vector2<> endPos = sp.getEndPosition();

        double relElong = sp.getRelElongation();
        double k = sp.getK();
    }
}
```

```
        continue;

    double color = relElong * k / 10.0;
    if (color < -1.0)
        color = -1.0;
    else if (color > 1.0)
        color = 1.0;

    if (color < 0.0)
    {
        glColor3d(-color / 2.0 + 0.5, 0.5 + color / 2.0,
                  0.5 + color / 2.0);
    }
    else
    {
        double green = 1.0 - 1.0 / (2.0 + 2.0 * color);
        glColor3d(1.0 - green, green, 1.0 - green);
    }

    glVertex2d(startPos.x, startPos.y);
    glVertex2d(endPos.x, endPos.y);
}
glEnd();

// draws all masses

glEnable(GL_POINT_SMOOTH);
glPointSize(3.0);
 glBegin(GL_POINTS);

glColor3f(0, 0.1, 0.5);
if (status==SHOW_ALL)
for (size_t i = 0; i < sms_.getNumMasses(); i++)
{
    MassProxy mp = sms_.getMassProxy(i);
    Vector2<> massPos = mp.getPos();
    glVertex2d(massPos.x, massPos.y);
}

glEnd();

glColor4d(1.0, 1.0, 1.0, 1.0);
unsigned it;

//IMPRIME LAS LONAS
vector<Element*> lonas = (sms_.getElementManager())->getLonas();

glBindTexture( GL_TEXTURE_2D, video_.textures.at(CINTA) );
glBegin(GL_QUADS);
if (status==SHOW_TEXT)
for (it=0; it<lonas.size();it++){
    for (size_t i = 0; i < lonas.at(it)->masas.size()-1; i++)
    {
        MassProxy mp = sms_.getMassProxy(lonas.at(it)->masas.at(i));
        Vector2<> massPos = mp.getPos();
        MassProxy mp2 = sms_.getMassProxy(lonas.at(it)->masas.at(i+1));
        Vector2<> massPos2 = mp2.getPos();
        double ang;
        double lseg = sqrt( pow(massPos.x-massPos2.x,2) + pow(massPos.y-
massPos2.y,2) );
        double dx,dy;
        ang = atan(ANCHO / lseg);
        dx = sin (ang) * ANCHO;
        dy = cos (ang) * ANCHO;

        glTexCoord2i( 0, 0 );
        glVertex3f( massPos.x, massPos.y, 0.0f );
        glTexCoord2i( 1, 0 );
```

```
    glVertex3f( massPos.x + dx, massPos.y - dy, 0.0f );
    glTexCoord2i( 1, 1 );
    glVertex3f( massPos2.x + dx, massPos2.y - dy, 0.0f );
    glTexCoord2i( 0, 1 );
    glVertex3f( massPos2.x, massPos2.y, 0.0f );
}
glEnd();

//IMPRIME LAS BARRAS DE METAL
vector<Element*> metalBars = (sms_.getElementManager())->getMetalBars();
glBindTexture( GL_TEXTURE_2D, video_.textures.at(BARRA) );
glBegin(GL_QUADS);
if (status==SHOW_TEXT)
for (it=0; it<metalBars.size();it++){
    MassProxy mp = sms_.getMassProxy(metalBars.at(it)->masas.at(0));
    Vector2<> massPos = mp.getPos();
    MassProxy mp2 = sms_.getMassProxy(metalBars.at(it)->masas.at(metalBars.at(it)->masas.size()/2-1));
    Vector2<> massPos2 = mp2.getPos();
    double ang;
    double lseg = sqrt( pow(massPos.x-massPos2.x,2) + pow(massPos.y-massPos2.y,2) );
    ang = acos( (massPos2.x-massPos.x) / lseg );
    double aX, aY,bX,bY,cX,cY,dX,dY;
    if ( massPos2.y < massPos.y )
    {
        if (massPos2.x<massPos.x)
            ang =ang + 2*(PI-ang);
        else
            ang = 2*PI - ang;
    }
    aX = massPos.x;
    aY = massPos.y;
    bX = (lseg)
        *cos(ang) + aX;
    bY = (lseg)
        *sin(ang) + aY;
    cX = (ANCH0)*sin(ang)
        + aX;
    cY = aY - (ANCH0)*cos(ang);
    dX = cX + (lseg)
        *cos(ang);
    dY = cY + (lseg)
        *sin(ang);

    glTexCoord2i( 0, 0 );
    glVertex3f(cX, cY, 0.0f);
    glTexCoord2i( 1, 0 );
    glVertex3f(aX, aY, 0.0f);
    glTexCoord2i( 1, 1 );
    glVertex3f(bX, bY, 0.0f);
    glTexCoord2i( 0, 1 );
    glVertex3f(dX, dY, 0.0f);
}
glEnd();

//IMPRIME LAS PLATAFORMAS
vector<Element*> plataformas = (sms_.getElementManager())->getPlataformas();

glBindTexture( GL_TEXTURE_2D, video_.textures.at(PLATAFORMA) );
glBegin(GL_QUADS);
if (status==SHOW_TEXT)
for (it=0; it<plataformas.size();it++){
    MassProxy mp = sms_.getMassProxy(plataformas.at(it)->masas.at(0));
    Vector2<> massPos = mp.getPos();
    MassProxy mp2 = sms_.getMassProxy(plataformas.at(it)->masas.at(plataformas.at(it)-174)
```

```
>masas.size()/2-1));
    Vector2<> massPos2 = mp2.getPos();
    double ang;
    double lseg = sqrt( pow(massPos.x-massPos2.x,2) + pow(massPos.y-
massPos2.y,2) );
    ang = acos( (massPos2.x-massPos.x) / lseg);
    double aX, aY, bX, bY, cX, cY, dX, dY;
    if ( massPos2.y < massPos.y)
    {
        if (massPos2.x<massPos.x )
            ang =ang + 2*(PI-ang);
        else
            ang = 2*PI - ang;
    }
    aX = massPos.x;
    aY = massPos.y;
    bX = (lseg)
        *cos(ang) + aX;
    bY = (lseg)
        *sin(ang) + aY;
    cX = (ANCHO)*sin(ang)
        + aX;
    cY = aY - (ANCHO)*cos(ang);
    dX = cX + (lseg)
        *cos(ang);
    dY = cY + (lseg)
        *sin(ang);

    glTexCoord2i( 0, 0);
    glVertex3f(cX, cY, 0.0f);
    glTexCoord2i( 1, 0);
    glVertex3f(aX, aY, 0.0f);
    glTexCoord2i( 1, 1);
    glVertex3f(bX, bY, 0.0f);
    glTexCoord2i( 0, 1);
    glVertex3f(dX, dY, 0.0f);
}
glEnd();

//IMPRIME LAS SOGAS
vector<Element*> ropes = (sms_.getElementManager())->getRopes();

glBindTexture( GL_TEXTURE_2D, video_.textures.at(SOGA) );
glBegin(GL_QUADS);
if (status==SHOW_TEXT)
for (it=0; it<ropes.size();it++){
    for (size_t i = 0; i < ropes.at(it)->masas.size()-1; i++)
    {
        MassProxy mp = sms_.getMassProxy(ropes.at(it)->masas.at(i));
        Vector2<> massPos = mp.getPos();
        MassProxy mp2 = sms_.getMassProxy(ropes.at(it)->masas.at(i+1));
        Vector2<> massPos2 = mp2.getPos();

        if ( abs(massPos.x - massPos2.x) < 0.009){
            glTexCoord2i( 0, 0 );
            glVertex3f( massPos.x, massPos.y, 0.0f );
            glTexCoord2i( 1, 0 );
            glVertex3f( massPos2.x, massPos2.y-0.02, 0.0f );
            glTexCoord2i( 1, 1 );
            glVertex3f( massPos2.x+0.005, massPos2.y, 0.0f );
            glTexCoord2i( 0, 1 );
            glVertex3f( massPos.x+0.005, massPos.y, 0.0f );
        }
    else {
        glTexCoord2i( 0, 0 );
        glVertex3f( massPos.x, massPos.y, 0.0f );
        glTexCoord2i( 1, 0 );
        glVertex3f( massPos.x, massPos.y-0.02, 0.0f );
    }
}
```

```

        glTexCoord2i( 1, 1 );
        glVertex3f( massPos2.x, massPos2.y-0.02, 0.0f );
        glTexCoord2i( 0, 1 );
        glVertex3f( massPos2.x, massPos2.y, 0.0f );
    }

}

glEnd();

//IMPRIME LAS BOLAS

vector<Element*> balls = (sms_.getElementManager())->getMainBalls();

if (status==SHOW_TEXT)
for (it=0; it<balls.size();it++){
    size_t cant = balls.at(it)->getLast() - balls.at(it)->getFirst
();

    MassProxy central = sms_.getMassProxy(balls.at(it)->getFirst());
    Vector2<> cenPos = central.getPos();

    glBindTexture( GL_TEXTURE_2D, video_.textures.at
(MASA) );

    glBegin(GL_QUADS);
    for (unsigned j = 1; j < cant; j++){
        MassProxy mp = sms_.getMassProxy(balls.at(it)->getFirst() + j);
        Vector2<> massPos = mp.getPos();
        MassProxy mp2 = sms_.getMassProxy(balls.at(it)->getFirst() + j + 1);
        Vector2<> massPos2 = mp2.getPos();

        glTexCoord2i( 0, 0 );
        glVertex3f(massPos.x, massPos.y, 0.0f );
        glTexCoord2i( 1, 0 );
        glVertex3f(cenPos.x, cenPos.y, 0.0f );

        glTexCoord2i( 1, 1 );
        glVertex3f(cenPos.x, cenPos.y, 0.0f );

        glTexCoord2i( 0, 1 );
        glVertex3f(massPos2.x, massPos2.y,
0.0f );
    }
    glEnd();
    glBindTexture( GL_TEXTURE_2D, video_.textures.at(9) );
    glBegin(GL_QUADS);
    MassProxy mp = sms_.getMassProxy(balls.at(it)->getLast());
    Vector2<> massPos = mp.getPos();
    MassProxy mp2 = sms_.getMassProxy(balls.at(it)->getFirst() + 1);
    Vector2<> massPos2 = mp2.getPos();

    glTexCoord2i( 0, 0 );
    glVertex3f(massPos.x, massPos.y, 0.0f );
    glTexCoord2i( 1, 0 );
    glVertex3f(cenPos.x, cenPos.y, 0.0f );

    glTexCoord2i( 1, 1 );
    glVertex3f(cenPos.x, cenPos.y, 0.0f );

    glTexCoord2i( 0, 1 );
    glVertex3f(massPos2.x, massPos2.y, 0.0f );

    glEnd();
}

//IMPRIME COHETES

vector<Element*> cohetes = (sms_.getElementManager())->getCohetes();

int cant;
int longitud;

```

```

glBindTexture( GL_TEXTURE_2D, video_.textures.at(COHETE) );
glBegin(GL_QUADS);
if (status==SHOW_TEXT)
    for (it=0; it<cohetes.size();it++){
        cant = cohetes.at(it)->masas.size() - 2;
        longitud = cant / 2;

        for (int i = 0; i < longitud-1; i++)
        {
            MassProxy mp = sms_.getMassProxy(cohetes.at(it)->masas.at(i));
            Vector2<> massPos = mp.getPos();
            MassProxy mp2 = sms_.getMassProxy(cohetes.at(it)->masas.at(i)+1);
            Vector2<> massPos2 = mp2.getPos();
            MassProxy mp3 = sms_.getMassProxy(cohetes.at(it)->masas.at(i)+longitud);
            Vector2<> massPos3 = mp3.getPos();
            MassProxy mp4 = sms_.getMassProxy(cohetes.at(it)->masas.at(i)+longitud+1);
            Vector2<> massPos4 = mp4.getPos();

            glTexCoord2i( 1, 0 );
            glVertex3f( massPos2.x, massPos2.y, 0.0f );
            glTexCoord2i( 1, 1 );
            glVertex3f( massPos.x, massPos.y, 0.0f );
            glTexCoord2i( 0, 1 );
            glVertex3f( massPos3.x, massPos3.y , 0.0f );
            glTexCoord2i( 0, 0 );
            glVertex3f( massPos4.x, massPos4.y, 0.0f );

        }
        MassProxy mp = sms_.getMassProxy(cohetes.at(it)->masas.at(cohetes.at(it)->masas.size
() - 2));
        Vector2<> massPos = mp.getPos();
        MassProxy mp2 = sms_.getMassProxy(cohetes.at(it)->masas.at(longitud-1));
        Vector2<> massPos2 = mp2.getPos();
        MassProxy mp3 = sms_.getMassProxy(cohetes.at(it)->masas.at(cohetes.at(it)->masas.size
() - 3));
        Vector2<> massPos3 = mp3.getPos();

        glTexCoord2i( 1, 0 );
        glVertex3f( massPos.x, massPos.y, 0.0f );
        glTexCoord2i( 1, 1 );
        glVertex3f( massPos2.x, massPos2.y, 0.0f );
        glTexCoord2i( 0, 1 );
        glVertex3f( massPos3.x, massPos3.y , 0.0f );
        glTexCoord2i( 0, 0 );
        glVertex3f( massPos.x, massPos.y, 0.0f );

    }

    glEnd();
}

//IMPRIME RUEDAS
vector<Element*> ruedas = (sms_.getElementManager())->getRuedas();
glBindTexture( GL_TEXTURE_2D, video_.textures.at(RUEDA) );
glBegin(GL_QUADS);
if (status==SHOW_TEXT)
    for (it=0; it<ruedas.size();it++){
        MassProxy central = sms_.getMassProxy(ruedas.at(it)->getFirst());
        Vector2<> cenPos = central.getPos();

        for (unsigned j = 1; j < ruedas.at(it)->masas.size()-1; j++){
            MassProxy mp = sms_.getMassProxy(ruedas.at(it)->masas.at(j));
            Vector2<> massPos = mp.getPos();
            MassProxy mp2 = sms_.getMassProxy(ruedas.at(it)->masas.at(j+1));
            Vector2<> massPos2 = mp2.getPos();

            glTexCoord2i( 0, 0 );
            glVertex3f(massPos.x, massPos.y, 0.0f );
            glTexCoord2i( 0, 1 );
            glVertex3f(cenPos.x, cenPos.y, 0.0f );
        }
    }
}

```

```

        glTexCoord2i( 1, 1 );
        glVertex3f(cenPos.x, cenPos.y, 0.0f );

        glTexCoord2i( 1, 0 );
        glVertex3f(massPos2.x, massPos2.y,
0.0f );
    }

    MassProxy mp = sms_.getMassProxy(ruedas.at(it)->masas.at(ruedas.at(it)-
>masas.size()-1));
    Vector2<> massPos = mp.getPos();
    MassProxy mp2 = sms_.getMassProxy(ruedas.at(it)->masas.at(1));
    Vector2<> massPos2 = mp2.getPos();

    glTexCoord2i( 0, 0 );
    glVertex3f(massPos.x, massPos.y, 0.0f );
    glTexCoord2i( 0, 1 );
    glVertex3f(cenPos.x, cenPos.y, 0.0f );

    glTexCoord2i( 1, 1 );
    glVertex3f(cenPos.x, cenPos.y, 0.0f );

    glTexCoord2i( 1, 0 );
    glVertex3f(massPos2.x, massPos2.y, 0.0f );
}

glEnd();

//DIBUJO PUNTOS FIJOS
glBindTexture( GL_TEXTURE_2D, video_.textures.at(PUNTO_FIJO) );
if (status==SHOW_TEXT)
for (size_t i = 0; i < sms_.getNumMasses(); i++)
{
    MassProxy mp = sms_.getMassProxy(i);
    Vector2<> massPos = mp.getPos();
    double invMass = mp.getInvMass();
    double massSize;
    massSize = MASS_SIZE/2;

    glBegin(GL_TRIANGLE_FAN);
    if (invMass==0)

        for (int j = 0; j <= POLY_SIDE; j++){
            glTexCoord2i( 0, 0 );
            glVertex3f(massPos.x, massPos.y, 0.0f );
            glTexCoord2i( 1, 0 );
            glVertex3f(massPos.x, massPos.y, 0.0f );

            glTexCoord2i( 1, 1 );
            glVertex3f(
                massPos.x + massSize *1.4* cos(j * (2 * PI / POLY_SIDE)),
                massPos.y + massSize *1.4* sin(j * (2 * PI / POLY_SIDE)), 0.0f );

            j++;
            glTexCoord2i( 0, 1 );
            glVertex3f(
                massPos.x + massSize *1.4* cos(j * (2 * PI / POLY_SIDE)),
                massPos.y + massSize *1.4* sin(j * (2 * PI / POLY_SIDE)), 0.0f );
        }
    }

    glEnd();
}
glEnd();

//INDICA TRANSPARENCIAS PARA AREA DE LLEGADA
glColor4d(1.0, 1.0, 4.0, 0.4);
 glEnable (GL_BLEND);
 glBlendFunc (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

//DIBUJO AREAS DE LLEGADA

```

```
vector<ZonaLlegada*> llegadas = (sms_.getElementManager())->getZonasLlegada();
ZonaLlegada* zona;
glBindTexture( GL_TEXTURE_2D, video_.textures.at(8) );
glBegin(GL_QUADS);

for (size_t i= 0 ; i<llegadas.size(); i++)
{
zona = llegadas.at(i);
glTexCoord2i( 0, 0 );
glVertex3f( zona->getX(), zona->getY(), 0.0f );
glTexCoord2i( 1, 0 );
glVertex3f( zona->getX(), zona->getY()-zona->getHeighth(), 0.0f );
glTexCoord2i( 1, 1 );
glVertex3f( zona->getX() + zona->getWidth(), zona->getY()-zona->getHeighth(),0.0f );
glTexCoord2i( 0, 1 );
glVertex3f( zona->getX() + zona->getWidth(), zona->getY(), 0.0f );

}

glEnd();

SDL_GL_SwapBuffers();
glColor4d(1.0, 1.0, 1.0, 1.0);
}
```

```
#include "Soga.h"

Soga::Soga() :
    ElementoLongitudinal("Longitudinal:Soga", Coordenada(long(10), long(10)), 25, 0, 10) {}

Soga::Soga(Cordenada posicion, ulong magnitud, long angulo, long ancho) :
    ElementoLongitudinal("Longitudinal:Soga", posicion, magnitud, angulo, ancho) {}

Soga::~Soga()
{}
```

```
#include "spring_mass_system.h"
#include <cassert>

namespace
{
    double calcDist(const Vector2<>& v1, const Vector2<>& v2)
    {
        Vector2<> d = v2;
        d -= v1;
        return d.norm();
    }

    void inelasticallyCollide(std::vector<Vector2<>>& y, SMSStaticState& x,
                               int m1, int m2, double distance)
    {
        // gets projections of velocity
        Vector2<> axis = y[m2];
        axis -= y[m1];
        double dist = axis.norm();
        axis.normalize();
        double v1p = axis.dot(y[m1 + y.size() / 2]);
        double v2p = axis.dot(y[m2 + y.size() / 2]);

        // checks if the masses really collide
        if (v1p < v2p)
            return;

        // m1 * vf + m2 * vf = m1 * v01 + m2 * v02
        // (1 / m2) * vf + (1 / m1) * vf = (1 / m2) * v01 + (1 / m1) * v02
        // vf = ((1 / m2) * v01 + (1 / m1) * v02) / ((1 / m1) + (1 / m2))
        double vf = (v1p * x.invMassVec[m2] + v2p * x.invMassVec[m1]) /
                     (x.invMassVec[m1] + x.invMassVec[m2]);

        // updates velocities
        Vector2<> dv1 = axis;
        Vector2<> dv2 = axis;
        dv1 *= (vf - v1p);
        dv2 *= (vf - v2p);
        y[m1 + y.size() / 2] += dv1;
        y[m2 + y.size() / 2] += dv2;

        // restores distance
        if (dist < distance)
        {
            Vector2<> dx = axis;
            dx *= (distance - dist) / 2.0;
            y[m1] -= dx;
            y[m2] += dx;
        }
    }

    void handleCollisions(std::vector<Vector2<>>& y, SMSStaticState& x)
    {
        // handles self collisions
        for (size_t i = 0; i < x.collMassesIndices.size(); i++)
        {
            for (size_t j = 0; j < i; j++)
            {
                int m1 = x.collMassesIndices[i];
                int m2 = x.collMassesIndices[j];
                double r1 = x.collMassesRadii[i];
                double r2 = x.collMassesRadii[j];

                if (calcDist(y[m1], y[m2]) >= r1 + r2)
                    continue;

                inelasticallyCollide(y, x, m1, m2, r1 + r2);
            }
        }
    }
}
```

```

void calcYPrime(std::vector<Vector2<>> & yPrimeVec,
    const std::vector<Vector2<>> & y,
    SMSStaticState& x, std::vector<Element*> cohetes, std::vector<Element*> ruedas,
    HiloCron *cronometro, long rocketTime)
{
    // spring forces
    for (size_t i = 0; i < x.springEndPointsVec.size(); i++)
    {
        Vector2<> delta(y[x.springEndPointsVec[i].second]);
        delta -= y[x.springEndPointsVec[i].first];

        double forceMag = - x.springKsVec[i] *
            (delta.norm() - x.springNormalLengthsVec[i]);

        // quick hack...
        if (fabs(forceMag) * x.springInvStrength[i] > 1.0)
            x.springKsVec[i] = 0.0;

        delta.normalize();
        delta *= forceMag;
        yPrimeVec[x.springEndPointsVec[i].second + y.size() / 2] += delta;

        delta *= -1.0;
        yPrimeVec[x.springEndPointsVec[i].first + y.size() / 2] += delta;
    }

    // viscous drag
    for (size_t i = 0; i < x.invMassVec.size(); i++)
    {
        Vector2<> drag(y[i + y.size() / 2]);
        drag *= x.damping;
        yPrimeVec[i + y.size() / 2] -= drag;
    }

    // dx/dt = v
    for (size_t i = 0; i < yPrimeVec.size() / 2; i++)
        yPrimeVec[i] = y[i + y.size() / 2];

    // dv/dt = force * (1 / mass)
    for (size_t i = 0; i < x.invMassVec.size(); i++)
        yPrimeVec[i + yPrimeVec.size() / 2] *= x.invMassVec[i];

    // adds gravity
    for (size_t i = 0; i < x.invMassVec.size(); i++)
        if (x.invMassVec[i] != 0.0)
            yPrimeVec[i + yPrimeVec.size() / 2] += x.g;

    unsigned it;
    double force, hip, fx, fy, dx, dy;
    force = 1.5;
    if (cronometro)
        if ((cronometro->getMilisegundos() <= rocketTime) || !rocketTime)
            std::cout << "vale" << cronometro->getMilisegundos() << std::endl;
    //}

    for (it=0; it<cohetes.size(); it++){
        // para cada una de los cohetes
        Vector2<> m1 = y[cohetes.at(it)->masas.at(cohetes.at(it)->masas.size()-1)];//
        mp.getPos();
        Vector2<> m2 = y[cohetes.at(it)->masas.at(cohetes.at(it)->masas.size()-2)];//
        mp.getPos();
        dx = m2.x - m1.x;
        dy = m2.y - m1.y;
        hip = sqrt(pow(dx,2) + pow(dy,2));
        fx = force * (dx/hip);
        fy = force * (dy/hip);

        for (unsigned i = 0; i < cohetes.at(it)->masas.size(); i+
        +)
            yPrimeVec[cohetes.at(it)->masas.at(i) + yPrimeVec.size() / 2] +=
        Vector2<> (fx,fy); //x.g;
    }
}

```

```
force = 1.5;
double alfa;
for (it=0; it<ruedas.size();it++){ //para cada una de los cohetes
    Vector2<> central = y[ruedas.at(it)->getFirst()];
    Vector2<> m2;
    double thisforce;
    for (unsigned i = ruedas.at(it)->getFirst() + 1; i <= ruedas.at(it)->getLast()
();i++){
        m2 = y[i];
        dx = m2.x - central.x;
        dy = m2.y - central.y;
        alfa = atan(dy/dx);
        thisforce = force * x.invMassVec[i];
        fx = thisforce * cos (alfa);
        fy = thisforce * sin (alfa);

        if (dy < 0) fy = -fy;
        yPrimeVec[i + yPrimeVec.size() / 2] += Vector2<> (fx,fy);
    }
}

}

MassProxy::MassProxy(SpringMassSystem& sms, size_t index) :
SMS_(sms), index_(index)
{
}

double MassProxy::getInvMass() const
{
    return SMS_.x_.invMassVec[index_];
}

const Vector2<>& MassProxy::getPos() const
{
    return SMS_.y_[index_];
}

const Vector2<>& MassProxy::getVel() const
{
    return SMS_.y_[index_ + SMS_.y_.size() / 2];
}

void MassProxy::setVel(const Vector2<>& v)
{
    SMS_.y_[index_ + SMS_.y_.size() / 2] = v;
}

SpringProxy::SpringProxy(SpringMassSystem& sms, size_t index) :
SMS_(sms), index_(index)
{
}

const Vector2<>& SpringProxy::getStartPos() const
{
    return SMS_.y_[SMS_.x_.springEndPointsVec[index_].first];
}

const Vector2<>& SpringProxy::getEndPos() const
{
    return SMS_.y_[SMS_.x_.springEndPointsVec[index_].second];
}

double SpringProxy::getRelElongation() const
{
    Vector2<> delta = getEndPos();
    delta -= getStartPos();
```

```
        return delta.norm() /
            SMS_.x_.springNormalLengthsVec[index_] - 1.0; //estos en 0
    }

double SpringProxy::getK() const
{
    return SMS_.x_.springKsVec[index_];
}

SpringMassSystem::SpringMassSystem(const Config& config)
{
    this->cronometro = NULL;
    x_.g = config.read<Vector2<>>("SMS.G");
    x_.damping = config.read<double>("SMS.Damping");
}

void SpringMassSystem::setCronometro(HiloCron *cronometro){
    this->cronometro = cronometro;
}
int SpringMassSystem::addMass(double invMass, const Vector2<>& pos)
{
    x_.invMassVec.push_back(invMass);
    y_.resize(y_.size() + 2);
    y_[y_.size() / 2 - 1] = pos;

    return x_.invMassVec.size() - 1;
}

int SpringMassSystem::addCollMass(double invMass, const Vector2<>& pos,
                                    double radius)
{
    int i = addMass(invMass, pos);
    x_.collMassesIndices.push_back(i);
    x_.collMassesRadii.push_back(radius);
    return i;
}

void SpringMassSystem::fixMass(int pos)
{
    x_.invMassVec.at(pos)=0;
}

int SpringMassSystem::addSpring(int startMass, int endMass, double k,
                                double relElongation, double
invStrength)
{
    x_.springEndPointsVec.push_back(
        std::make_pair(startMass, endMass));
    x_.springKsVec.push_back(k);

    Vector2<> delta(y_[endMass]);
    delta -= y_[startMass];

    x_.springNormalLengthsVec.push_back(
        delta.norm() / (1.0 + relElongation));

    x_.springInvStrength.push_back(invStrength);

    return x_.springKsVec.size() - 1;
}

size_t SpringMassSystem::getNumMasses() const
{
    return x_.invMassVec.size();
}

size_t SpringMassSystem::getNumSprings() const
{
    return x_.springKsVec.size();
}

MassProxy SpringMassSystem::getMassProxy(size_t index)
```

```
{  
    return MassProxy(*this, index);  
}  
  
SpringProxy SpringMassSystem::getSpringProxy(size_t index)  
{  
    return SpringProxy(*this, index);  
}  
  
void SpringMassSystem::setMapa(Mapa* mapa){  
    this->mapa = mapa;  
}  
  
Mapa* SpringMassSystem::getMapa(){  
    return this->mapa;  
}  
  
void SpringMassSystem::unifyMass(int first,int second){  
    for (unsigned i=0; i<manager.vectoresElementos.size();i++) //en los elementos reemplaza las  
masas unificadas  
        for (unsigned j=0; j<manager.vectoresElementos.at(i)->size();j++)  
            for (unsigned k=0; k<(manager.vectoresElementos.at(i)->at(j))->masas.size();k+  
+){  
                if ( ((manager.vectoresElementos.at(i)->at(j))->masas.at(k) ) ==  
first ){  
                    ((manager.vectoresElementos.at(i)->at(j))->masas.at(k) ) =  
second;  
                }  
            }  
    }  
    for (unsigned i=0; i<x_.springEndPointsVec.size();i++){  
        if (x_.springEndPointsVec.at(i).first == first) {  
            x_.springEndPointsVec.at(i).first = second;  
        }  
        if (x_.springEndPointsVec.at(i).second == first) {  
            x_.springEndPointsVec.at(i).second = second;  
        }  
    }  
}  
std::vector<int> SpringMassSystem::getVecEnganches(){  
    return enganches;  
}  
  
void SpringMassSystem::addEnganche(int enganche){  
    enganches.push_back(enganche);  
}  
  
bool estanCerca(Vector2<> pos1,Vector2<> pos2){  
    double modulo,dx,dy;  
    dx = pos1.x - pos2.x;  
    dy = pos2.y - pos1.y;  
  
    modulo = sqrt(pow(dx,2)+pow(dy,2));  
    return (modulo <= 0.04);  
}  
  
void SpringMassSystem::setRocketTime(long time){  
    rocketTime = time;  
}  
  
void SpringMassSystem::checkRoutine(){  
    std::vector<int> enganches = getVecEnganches();  
  
    for (size_t i = 0; i< enganches.size() - 1; i++){  
        for (size_t j=i+1; j<enganches.size(); j++){  
            MassProxy mp2 = getMassProxy(enganches.at(j));  
            Vector2<> pos2 = mp2.getPos();  
            MassProxy mp = getMassProxy(enganches.at(i));  
            Vector2<> pos1 = mp.getPos();  
    }
```

```
        if (estanCerca(pos1,pos2)){
            if (mp.getInvMass() != 0){
                unifyMass(enganches.at(i),enganches.at(j));
                y_.at(enganches.at(j)).x=y_.at(enganches.at
(i)).x;
                y_.at(enganches.at(j)).y=y_.at(enganches.at(i)).y;
                y_.at(enganches.at(i)).x=rand()*12;
                y_.at(enganches.at(i)).y=5;
            }
            else {
                unifyMass(enganches.at(j),enganches.at(i));
                y_.at(enganches.at(i)).x=y_.at(enganches.at
(j)).x;
                y_.at(enganches.at(i)).y=y_.at(enganches.at(j)).y;
                y_.at(enganches.at(j)).x=rand()*12;
                y_.at(enganches.at
(j)).y=5;
            }
        }
    }

int mas1, mas2;

for (size_t i = 0; i< x_.springNormalLengthsVec.size();i++){
    mas1 = x_.springEndPointsVec.at(i).first;
    mas2 = x_.springEndPointsVec.at(i).second;

    Vector2<> delta(y_[mas2]);
    delta -= y_[mas1];

    x_.springNormalLengthsVec[i] = (delta.norm());
}
}

double SpringMassSystem::getPotentialEnergy() const
{
    double pe = 0.0;

    Vector2<> deltaVec;
    double deltaNorm;

    // accumulates springs' potential energy
    for (size_t i = 0; i < x_.springEndPointsVec.size(); i++)
    {
        deltaVec = y_[x_.springEndPointsVec[i].second];
        deltaVec -= y_[x_.springEndPointsVec[i].first];
        deltaNorm = deltaVec.norm() - x_.springNormalLengthsVec[i];
        pe += 0.5 * x_.springKsVec[i] * deltaNorm * deltaNorm;
    }

    // accumulates masses' potential energy
    for (size_t i = 0; i < x_.invMassVec.size(); i++)
        if (x_.invMassVec[i] != 0.0)
            pe += -y_[i].dot(x_.g) / x_.invMassVec[i];
}

return pe;
}

double SpringMassSystem::getKineticEnergy() const
{
    double ke = 0.0;

    // accumulates masses' kinetic energy
    for (size_t i = 0; i < y_.size() / 2; i++)
        if (x_.invMassVec[i] != 0.0)
            ke += (0.5 / x_.invMassVec[i]) * y_[i + y_.size() / 2].sqNorm();
        else
            continue;

    return ke;
}
```

```
double SpringMassSystem::getEnergy() const
{
    return getPotentialEnergy() + getKineticEnergy();
}

void SpringMassSystem::eulerStep(double dt)
{
    std::vector<Vector2> > yPrimeVec(y_.size());
    // calculates derivative of system state (y prime)
    calcYPrime(yPrimeVec, y_, x_, manager.getCohetes(), manager.getRuedas(), cronometro, rocketTime);

    // update positions and velocities
    for (size_t i = 0; i < y_.size(); i++)
    {
        Vector2 delta = yPrimeVec[i];
        delta *= dt;
        y_[i] += delta;
    }

    // handles collisions
    handleCollisions(y_, x_);
}

void SpringMassSystem::rk4Step(double dt)
{
    std::vector<Vector2> > k1(y_.size());
    std::vector<Vector2> > k2(y_.size());
    std::vector<Vector2> > k3(y_.size());
    std::vector<Vector2> > k4(y_.size());

    std::vector<Vector2> > yTmp(y_.size());
    // gets k1
    calcYPrime(k1, y_, x_, manager.getCohetes(), manager.getRuedas(), cronometro, rocketTime);

    for (size_t i = 0; i < yTmp.size(); i++)
    {
        Vector2 delta = k1[i];
        delta *= dt / 2.0;
        yTmp[i] = y_[i];
        yTmp[i] += delta;
    }

    // gets k2
    calcYPrime(k2, yTmp, x_, manager.getCohetes(), manager.getRuedas(), cronometro, rocketTime);

    // gets argument to calculate k3
    for (size_t i = 0; i < yTmp.size(); i++)
    {
        Vector2 delta = k2[i];
        delta *= dt / 2.0;
        yTmp[i] = y_[i];
        yTmp[i] += delta;
    }

    // gets k3
    calcYPrime(k3, yTmp, x_, manager.getCohetes(), manager.getRuedas(), cronometro, rocketTime);

    // gets argument to calculate k4
    for (size_t i = 0; i < yTmp.size(); i++)
    {
        Vector2 delta = k3[i];
        delta *= dt;
        yTmp[i] = y_[i];
        yTmp[i] += delta;
    }

    // gets k4
    calcYPrime(k4, yTmp, x_, manager.getCohetes(), manager.getRuedas(), cronometro, rocketTime);

    // updates state
}
```

```
for (size_t i = 0; i < y_.size(); i++)
{
    Vector2<> delta = k2[i];
    delta += k3[i];
    delta *= 2.0;
    delta += k1[i];
    delta += k4[i];

    delta *= dt / 6.0;

    y_[i] += delta;
}

// handles collisions
handleCollisions(y_, x_);

void SpringMassSystem::debugPrint(std::ostream& os) const
{
    for (size_t i = 0; i < x_.invMassVec.size(); i++)
    {
        os << "Mass " << i << "\n";
        os << "\tInvMass: " << x_.invMassVec[i] << "\n";
        os << "\tPos: " << y_[i] << "\n";
        os << "\tVel: " << y_[i + y_.size() / 2] << "\n";
    }

    for (size_t i = 0; i < x_.springEndPointsVec.size(); i++)
    {
        os << "Spring " << i << "\n";
        os << "\tStart point: " << x_.springEndPointsVec[i].first << "\n";
        os << "\tEnd point: " << x_.springEndPointsVec[i].second << "\n";
        os << "\tK: " << x_.springKsVec[i] << "\n";
        os << "\tNormalLength: " << x_.springNormalLengthsVec[i] << "\n";
    }
}

ElementManager* SpringMassSystem::getElementManager(){
    return &manager;
}
```

```
#include "texture_store.h"
#include <SDL.h>
#include <SDL_image.h>

TextureStore::TTexStore TextureStore::texStore_;

namespace
{
    static void do_loadtexture(const char * aFilename, int clamp = 1)
    {
        int i, j;

        // Load texture using SDL_Image
        SDL_Surface *temp = IMG_Load(aFilename);

        if (temp == NULL)
            return;

        // Set up opengl-compatible pixel format
        SDL_PixelFormat pf;
        pf.palette = NULL;
        pf.BitsPerPixel = 32;
        pf.BytesPerPixel = 4;
        pf.alpha = 0;
        pf.colorkey = 0;
#if SDL_BYTEORDER == SDL_LIL_ENDIAN
        pf.Rmask = 0x000000ff;
        pf.Rshift = 0;
        pf.Rloss = 0;
        pf.Gmask = 0x0000ff00;
        pf.Gshift = 8;
        pf.Gloss = 0;
        pf.Bmask = 0x00ff0000;
        pf.Bshift = 16;
        pf.Bloss = 0;
        pf.Amask = 0xff000000;
        pf.Ashift = 24;
        pf.Aloss = 0;
#else
        pf.Amask = 0x000000ff;
        pf.Ashift = 0;
        pf.Aloss = 0;
        pf.Bmask = 0x0000ff00;
        pf.Bshift = 8;
        pf.Bloss = 0;
        pf.Gmask = 0x00ff0000;
        pf.Gshift = 16;
        pf.Gloss = 0;
        pf.Rmask = 0xff000000;
        pf.Rshift = 24;
        pf.Rloss = 0;
#endif
        // Convert texture to said format
        SDL_Surface *tm = SDL_ConvertSurface(temp, &pf, SDL_SWSURFACE);

        // Cleanup
        SDL_FreeSurface(temp);

        // Lock the converted surface for reading
        SDL_LockSurface(tm);

        int w,h,l;
        w = tm->w;
        h = tm->h;
        l = 0;
        unsigned int * mip = new unsigned int[w * h * 5];
        unsigned int * src = (unsigned int *)tm->pixels;

        memset(mip, 0, w*h*4);

        // mark all pixels with alpha = 0 to black
        for (i = 0; i < h; i++)
            for (j = 0; j < w; j++)
                if ((src[i*w+j] & 0x000000ff) == 0)
                    mip[l] = 0;
                else
                    mip[l] = src[i*w+j];
                l++;
    }
}
```

```
{  
    for (j = 0; j < w; j++)  
    {  
        if ((src[i*w+j] & pf.Amask) == 0)  
            src[i*w+j] = 0;  
    }  
}  
  
// Tell OpenGL to read the texture  
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, tm->w, tm->h, 0, GL_RGBA, GL_UNSIGNED_BYTE,  
(GLvoid*)src);  
  
if (mip)  
{  
    // precalculate summed area tables  
    // it's a box filter, which isn't very good, but at least it's fast =)  
    int ra = 0, ga = 0, ba = 0, aa = 0;  
    int i, j, c;  
    unsigned int * rbuf = mip + (tm->w * tm->h * 1);  
    unsigned int * gbuf = mip + (tm->w * tm->h * 2);  
    unsigned int * bbuf = mip + (tm->w * tm->h * 3);  
    unsigned int * abuf = mip + (tm->w * tm->h * 4);  
  
    for (j = 0, c = 0; j < tm->h; j++)  
    {  
        ra = ga = ba = aa = 0;  
        for (i = 0; i < tm->w; i++, c++)  
        {  
            ra += (src[c] >> 0) & 0xff;  
            ga += (src[c] >> 8) & 0xff;  
            ba += (src[c] >> 16) & 0xff;  
            aa += (src[c] >> 24) & 0xff;  
            if (j == 0)  
            {  
                rbuf[c] = ra;  
                gbuf[c] = ga;  
                bbuf[c] = ba;  
                abuf[c] = aa;  
            }  
            else  
            {  
                rbuf[c] = ra + rbuf[c - tm->w];  
                gbuf[c] = ga + gbuf[c - tm->w];  
                bbuf[c] = ba + bbuf[c - tm->w];  
                abuf[c] = aa + abuf[c - tm->w];  
            }  
        }  
    }  
}  
  
while (w > 1 || h > 1)  
{  
    l++;  
    w /= 2;  
    h /= 2;  
    if (w == 0) w = 1;  
    if (h == 0) h = 1;  
  
    int dw = tm->w / w;  
    int dh = tm->h / h;  
  
    for (j = 0, c = 0; j < h; j++)  
    {  
        for (i = 0; i < w; i++, c++)  
        {  
            int x1 = i * dw;  
            int y1 = j * dh;  
            int x2 = x1 + dw - 1;  
            int y2 = y1 + dh - 1;  
            int div = (x2 - x1) * (y2 - y1);  
            y1 *= tm->w;  
            y2 *= tm->w;  
        }  
    }  
}
```

```

+ rbuf[y1 + x1];
+ gbuf[y1 + x1];
+ bbuf[y1 + x1];
+ abuf[y1 + x1];

        int r = rbuf[y2 + x2] - rbuf[y1 + x2] - rbuf[y2 + x1];
        int g = gbuf[y2 + x2] - gbuf[y1 + x2] - gbuf[y2 + x1];
        int b = bbuf[y2 + x2] - bbuf[y1 + x2] - bbuf[y2 + x1];
        int a = abuf[y2 + x2] - abuf[y1 + x2] - abuf[y2 + x1];

        r /= div;
        g /= div;
        b /= div;
        a /= div;

        if (a == 0)
            mip[c] = 0;
        else
            mip[c] = ((r & 0xff) << 0) |
                ((g & 0xff) << 8) |
                ((b & 0xff) << 16) |
                ((a & 0xff) << 24);
    }
    glTexImage2D(GL_TEXTURE_2D, l, GL_RGBA, w, h, 0, GL_RGBA,
GL_UNSIGNED_BYTE, (GLvoid*)mip);
}
glTexParameteri
(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR_MIPMAP_LINEAR); // Linear Filtering
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LINEAR); // Linear Filtering
delete[] mip;
}
else
{
    glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR); // Linear Filtering
    glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LINEAR); // Linear Filtering
}

// Unlock..
SDL_UnlockSurface(tm);

// and cleanup.
SDL_FreeSurface(tm);

if (clamp)
{
    // Set up texture parameters
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);
}
else
{
    // Set up texture parameters
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
}

GLuint load_texture(const char* aFilename, int clamp = 1)
{
    // Create OpenGL texture handle and bind it to use

    GLuint texname;
    glGenTextures(1,&texname);
    glBindTexture(GL_TEXTURE_2D,texname);

    do_loadtexture(aFilename);

    return texname;
}

```

```
    }

GLuint TextureStore::addTexture(const std::string& filename)
{
    TTexStore::iterator it = texStore_.find(filename);

    // checks if it's already stored
    if (it != texStore_.end())
        return it->second;

    // creates the texture
    GLuint texID = load_texture(filename.c_str());

    // stores the id
    texStore_[filename] = texID;

    // returns the ID
    return texID;
}
```

```
#include "Ventana.h"
#include <iostream>
#include "Boton.h"
#include <vector>
#include "InterfazMapa.h"
#include "ParserXML.h"
#include "sms_app.h"

using namespace std;

Ventana::Ventana(int ancho, int alto, ChatDoble* chat) {

    this->alto = alto;
    this->ancho = ancho;
    this->salir = false;
    this->sms = false;
    this->pideSimulacion = false;

    this->chat = chat;

    int w = ancho;
    int h = alto;
    int depth = 16;

    SDL_InitSubSystem(SDL_INIT_VIDEO);
    SDL_GL_SetAttribute(SDL_GL_DOUBLEBUFFER, 1);
    pantalla = SDL_SetVideoMode(w, h, depth, SDL_OPENGL | SDL_FULLSCREEN);
    SDL_WM_SetCaption("Editor de Escenarios", NULL);

    double ar = static_cast<double>(w) / h;
    glOrtho(-ar, ar, -1.0, 1.0, -1.0, 1.0);
    glEnable(GL_TEXTURE_2D);

    glClearColor( 0.0f, 0.0f, 0.0f, 0.0f);

    glClear(GL_COLOR_BUFFER_BIT);

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();

    glClearColor(1.0f, 1.0f, 1.0f, 0.0f);

    if (chat != NULL)
        chat->cargarTexturas();

    reloj.cargarTexturas();
    displayDinero.cargarTexturas();
    cronometro = new HiloCron();
}

Ventana::~Ventana() {

    delete cronometro;

    list<PanelContenedor*>::iterator it;
    for (it = _paneles.begin(); it != _paneles.end(); it++) {
        delete (*it);
    }

    SDL_QuitSubSystem(SDL_INIT_VIDEO);
    SDL_Quit();
}

void Ventana::ejecutar(void* cont, void* mapa) {
    SDL_Event eventos;
    Coordenada posicionMouse;

    SDL_EnableUNICODE(1);
```

```
string movimiento;

Mapa * miMapa = (Mapa*) mapa;
InterfazMapa* contrincante = (InterfazMapa*) cont;
ParserXML par;

bool esperar = false;

char letra;
int plataInicial = miMapa->getPlata();
cronometro->start();
while (!salir) {
    //*****IMPRIMIR RELOJ*****
    double tiempo = cronometro->getMilisegundos() / TIME_C;
    stringstream ss;

    ss << fixed;
    ss.precision( 2 );
    ss << tiempo;

    //*****FIN IMPRIMIR RELOJ*****
    string time = ss.str();
    time += " s.';

    stringstream ss2;

    ss2.precision( 0 );
    ss2 << miMapa->getPlata();
    string money = ss2.str();

money = "$" + money;
imprimir_pantalla(time, money);

if (pideSimulacion)
{
    if (chat != NULL)
    {
        chat->agregarMensaje("L");
        pideSimulacion = false;
        //esperar = true;
    }
    else
    {
        pideSimulacion = false;
        SMSApp app(sms, miMapa);
        double tiempoSimulacion = app.run();
        double tiempoResolucion = miMapa->getTiempoResolucion()/1000;
        if(tiempoSimulacion > 0.0) {
            int plataUsada = plataInicial - miMapa->getPlata();

            if(miMapa->getPlata() >= 0)
                miMapa->setPuntos(calcularPuntaje(CTE_A, CTE_B, CTE_C,
CTE_D,
plataUsada));
            else
                miMapa->setPuntos(0);
            salir = true;
            continue;
        }
    }
}

if (chat != NULL)
{
    if (chat->debeSalir())
        salir = true;
    if (chat->obtenerMovimiento(movimiento))
    {
        contrincante->setMapa(par.obtenerMapaMemoria(movimiento));
    }
}
```

```
if (chat->debeBloquear())
{
    esperar = true;
    chat->setBloquear(false);
}
if (chat->debeSimular())
{
    SMSApp app(sms, miMapa);
    double tiempoSimulacion = app.run();
    double tiempoResolucion = miMapa->getTiempoResolucion()/1000;
    if(tiempoSimulacion > 0.0) {
        int plataUsada = plataInicial - miMapa->getPlata();
        if(plataUsada >= 0)
            miMapa->setPuntos(calcularPuntaje(CTE_A, CTE_B, CTE_C,
CTE_D,
plataUsada));
        else
            miMapa->setPuntos(0);
        salir = true;
        continue;
    }
    else
    {
        chat->agregarMensaje("N");
    }
    chat->setSimular(false);
    esperar = false;
}

if (esperar) continue;
while (SDL_PollEvent(&eventos)) {
    posicionMouse.setX(eventos.motion.x);
    posicionMouse.setY(eventos.motion.y);
    switch (eventos.type) {
    case SDL_QUIT:
        salir = true;
        break;

    case SDL_MOUSEBUTTONDOWN:
        seEjecutaEventoEn(posicionMouse, SDL_MOUSEBUTTONDOWN);
        break;

    case SDL_MOUSEBUTTONUP:
        seEjecutaEventoEn(posicionMouse, SDL_MOUSEBUTTONUP);
        break;

    case SDL_MOUSEMOTION:
        seEjecutaEventoEn(posicionMouse, SDL_MOUSEMOTION);
        break;

    case SDL_KEYDOWN:
        if (chat == NULL)
            break;

        if (eventos.key.keysym.sym == SDLK_RETURN)
        {
            if(!chat->enter())
                cout << "buffer vacio" << endl;
            break;
        }

        if (eventos.key.keysym.sym == SDLK_TAB)
        {
            chat->cambiarVisible();
            break;
        }

        if (eventos.key.keysym.sym == SDLK_BACKSPACE)
```

```
        {
            chat->quitarLetra();
        }

        letra = chat->esLetra(eventos);
        if (letra == -1) break;
        chat->agregarLetra(letra);

    default:
        break;
    }
}

cronometro->terminar();
cronometro->join();
}

long Ventana::calcularPuntaje(long a, long b, long c, long d, long tS, long tR, int sumaPrecios) {
    return (a/(tR + b)) + (c/(tS + d)) - sumaPrecios;
}

void Ventana::agregarPanel(PanelContenedor * panel, const Coordenada posicion) {
    panel->setVentana(this);
    panel->setPantalla(pantalla);
    panel->setPosicion(posicion);
    panel->setId(_paneles.size() + 1);
    _paneles.push_back(panel);
}

void Ventana::agregarPanel(PanelContenedor * panel, const int lugar) {
    Coordenada pos;
    panel->setVentana(this);
    switch (lugar) {
    case 0:
        pos.setX(20 + _paneles.size() * (panel->getAncho() + 20));
        pos.setY(this->alto - panel->getAltura());
        break;
    case 1:
        pos.setX(0);
        pos.setY(this->alto - panel->getAltura());
    }

    panel->setPantalla(pantalla);
    panel->setPosicion(pos);
    panel->setId(_paneles.size() + 1);
    _paneles.push_back(panel);
}

void Ventana::imprimir_pantalla(string & tiempo, string & money) {

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    list<PanelContenedor*>::iterator it;
    for (it = _paneles.begin(); it != _paneles.end(); it++) {
        (*it)->imprimir();
    }
    reloj.imprimirReloj(tiempo, 1.25, -0.9);
    displayDinero.imprimirReloj(money, -1.7, 1);
    if (chat != NULL)
        chat->imprimirChat();

    SDL_GL_SwapBuffers();
}

bool Ventana::seEjecutaEventoEn(const Coordenada punto, int evento) {
    list<PanelContenedor*>::iterator it;
    for (it = _paneles.begin(); it != _paneles.end(); it++) {
        if ((*it)->pertenece(punto)) {
            if ((*it)->seEjecutaEventoEn(punto, evento)) {
                return false;
            }
        }
    }
}
```

```
        }

    }

    switch(evento) {
        case SDL_MOUSEBUTTONDOWN: evento_MouseDown(punto); break;
        case SDL_MOUSEMOTION: evento_MouseOver(punto); break;
        case SDL_MOUSEBUTTONUP: evento_MouseUp(punto); break;
    }

    return true;
}

void Ventana::evento_MouseDown(Cordenada punto) {

}

void Ventana::evento_MouseOver(Cordenada punto) {

    list<PanelContenedor*>::iterator it;
    for (it = _paneles.begin(); it != _paneles.end(); it++)
        if((*it)->getNombre().find("Boton") != string::npos) {
            Boton * boton = (Boton*) (*it);
            boton->setMouseOver(false);
            if(boton->esta_presionado())
                boton->setImagenDeFondo(TEXTURA_MOUSEDOWN);
            else
                boton->setImagenDeFondo(TEXTURA_DEFECTO);
        }
}

void Ventana::evento_MouseUp(Cordenada punto) {

}

void Ventana::desclickearBotones(int id_excepcion) {
    list<PanelContenedor*>::iterator it;
    for (it = _paneles.begin(); it != _paneles.end(); it++)
        if((*it)->getNombre().find("Boton") != string::npos) {
            Boton * boton = (Boton*) (*it);
            if(boton->getId() == id_excepcion) {
                continue;
            } else {
                boton->desapretar();
                boton->setImagenDeFondo(TEXTURA_DEFECTO);
            }
        }
}

void Ventana::corregirCoordenadas(ulong *x, ulong *y) {
    *x = (*x - (ancho/2.0)) / 300;
    *y = (-*y + (alto/2.0)) / 300;
}

long Ventana::getTiempoResolucion() {
    return cronometro->getMilisegundos();
}
```

```
#include "video.h"

void Video::loadSurface(const char* surf){
    GLuint texture;
    SDL_Surface *surface;
    GLenum texture_format=0;
    GLint nOfColors;
    if ( (surface = IMG_Load(surf)) ) {

        if ( (surface->w & (surface->w - 1)) != 0 ) {
            printf("ERROR en imagen, el ancho no es potencia de 2\n");
        }

        if ( (surface->h & (surface->h - 1)) != 0 ) {
            printf("ERROR en imagen, el alto no es potencia de 2\n");
        }

        nOfColors = surface->format->BytesPerPixel;
        if (nOfColors == 4)
        {
            if (surface->format->Rmask == 0x000000ff)
                texture_format = GL_RGBA;
            else
                texture_format = GL_BGRA;
        } else if (nOfColors == 3)
        {
            if (surface->format->Rmask == 0x000000ff)
                texture_format = GL_RGB;
            else
                texture_format = GL_BGR;
        } else {
            printf("Atención: la imagen no es truecolor\n");
        }
    }

    glGenTextures( 1, &texture );
    glBindTexture( GL_TEXTURE_2D, texture );

    glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR );
    glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR );
    glTexImage2D( GL_TEXTURE_2D, 0, nOfColors, surface->w, surface->h, 0,
                  texture_format, GL_UNSIGNED_BYTE, surface->pixels );
}
else {
    printf("No se pudo cargar la imagen: %s\n", SDL_GetError());
    SDL_Quit();
}

if ( surface ) {
    SDL_FreeSurface( surface );
}
textures.push_back(texture);
}

void Video::loadSurfaces(){
    loadSurface("ima/mapa/ball.bmp"); //0
    loadSurface("ima/mapa/punto_fijo.bmp"); //1
    loadSurface("ima/mapa/rocket.bmp"); //2
    loadSurface("ima/mapa/metal.bmp"); //3
    loadSurface("ima/mapa/platform.bmp"); //4
    loadSurface("ima/mapa/rope.bmp"); //5
    loadSurface("ima/mapa/lona.bmp"); //6
    loadSurface("ima/mapa/rocket.bmp"); //7
    loadSurface("ima/mapa/llegada.bmp"); //8
    loadSurface("ima/mapa/ballbright.bmp"); //9
    loadSurface(pathFondo); //10
    fondo = textures.at(10);
}
```

```
}

Video::Video(const Config& config, const char* pathFondo)
{
    this->pathFondo = pathFondo;
    int w = config.read<int>("Video.Width");
    int h = config.read<int>("Video.Height");
    int depth = config.read<int>("Video.Depth");

    SDL_InitSubSystem(SDL_INIT_VIDEO);
    SDL_GL_SetAttribute(SDL_GL_DOUBLEBUFFER, 1);
    SDL_SetVideoMode(w, h, depth, SDL_OPENGL | SDL_FULLSCREEN);

    SDL_WM_SetCaption("Tatu Carreta RUN", 0);

    double ar = static_cast<double>(w) / h;
    glOrtho(-ar, ar, -1.0, 1.0, -1.0, 1.0);
    glEnable( GL_TEXTURE_2D );

    glClearColor( 0.0f, 0.0f, 0.0f, 0.0f );
    glClear( GL_COLOR_BUFFER_BIT );
    glMatrixMode( GL_PROJECTION );
    glLoadIdentity();

    loadSurfaces();
    glClearColor(1.0f, 1.0f, 1.0f, 0.0f);
}
```

```
#include "zonaLlegada.h"

ZonaLlegada::ZonaLlegada(double x, double y, double width, double height){
    this->x = x;
    this->y = y;
    this->width = width;
    this->height = height;
}
double ZonaLlegada::getX(){
    return x;
}
double ZonaLlegada::getY(){
    return y;
}
double ZonaLlegada::getWidth(){
    return width;
}
double ZonaLlegada::getHeight(){
    return height;
}
```

Código Fuente

Servidor

```
#ifndef __MUTEX_H__
#define __MUTEX_H__

#include <pthread.h>

class Mutex
{
    private:
        pthread_mutex_t mut;

        /* Constructor copia privado */
        Mutex(const Mutex& m);

        /* Operador = privado */
        Mutex& operator = (const Mutex& original);

    public:
        /* Constructor */
        Mutex();

        /* Lockea el mutex */
        void lock();

        /* Deslockea el mutex */
        void unlock();
};

#endif
```

```
#ifndef __COMMON_SOCKET_H__
#define __COMMON_SOCKET_H__


#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <iostream>

#define SOCKET_INVALIDO -1
#define BACKLOG 10
#define TAM_BUFFER 256

class Socket
{

private:
    int id; //Identificador del Socket
    sockaddr_in dir; //Mi dirección
    char historial[TAM_BUFFER + 1]; //Bytes remanentes del recv
    unsigned int validosHistorial; //Cantidad de bytes remanentes del recv

public:
    /* Constructor */
    Socket();

    /* Destructor */
    virtual ~Socket();

    /* Crea al socket. Devuelve true si fue una creación exitosa. */
    bool create();

    /* Envía a través del Socket el dato indicado.
     * Devuelve true si pudo enviar correctamente */
    bool send(const std::string& dato) const;

    /* Recibe un mensaje a través del Socket y lo almacena en dato.
     * Devuelve true si pudo recibir correctamente */
    bool recv(std::string& dato);

    /* Modifica el Id del Socket */
    void setId(const int id);

    /* Devuelve el Id del Socket */
    int getId() const;

    /* Devuelve mi dirección */
    sockaddr_in& getDir();

    /* Cierra el Socket */
    void close();

    /* Shutdown del Socket */
    void shutdown(int modo);

};

#endif
```

```
#ifndef __SOCKET_CLIENTE_H__
#define __SOCKET_CLIENTE_H__


#include "common_Socket.h"
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <iostream>

class SocketCliente: public Socket
{
public:
    /* Trata de conectarse a un Socket que se encuentra escuchando
     * en la dirección y puerto indicados.
     * Devuelve true si fue una conexión exitosa. */
    bool connect(const std::string direccion, const int puerto);
};

#endif
```

```
#ifndef __THREAD_H__
#define __THREAD_H__

#include <pthread.h>
#include "common_Mutex.h"

class Thread
{
    private:
        pthread_t hilo;
        Mutex mHilo; //Evita que se lean y escriban los atributos al mismo tiempo.
        bool vivo; //True si tiene que seguir corriendo
        bool corriendo; //True si el hilo está corriendo

        /* Constructor copia privado */
        Thread(const Thread& t);

        /* Operador = privado */
        Thread& operator = (const Thread& original);

    public:
        /* Constructor */
        Thread();

        /* Destructor */
        virtual ~Thread();

        /* Le indica al hilo que debe suicidarse */
        void terminar();

        /* Devuelve true si el hilo aún está corriendo */
        bool estaCorriendo();

        /* Método estático que se empieza a correr al crearse el hilo de tipo pthread */
        static void* static_run (void* p);

        virtual void run() = 0;

        /* Espera que el hilo termine su ejecución */
        void join() const;

        /* Devuelve true si el hilo debe seguir vivo */
        bool estaVivo();

        /* Método que inicia la ejecución del hilo */
        void start();

    protected:
        /* Bloquear y desbloquear estado se utilizan para evitar que se acceda
         * a un atributo del hilo al mismo tiempo (desde el hilo mismo y desde el
         * hilo que lo creó */
        void bloquearEstado();
        void desbloquearEstado();

        /* Modifica el valor de corriendo */
        void setCorriendo(bool valor);

};

#endif
```

```
#ifndef __ADMINISTRADOR_USUARIOS_H__
#define __ADMINISTRADOR_USUARIOS_H__


#include <map>
#include <vector>
#include <string>
#include "common_Thread.h"
#include "server_HiloManejadorCliente.h"
#include "server_Mensaje.h"
#include "server_HiloBatalla.h"


#define MAX_PARTIDAS 10

class AdministradorUsuarios: public Thread
{
    private:

        /* Cantidad de partidas que se están jugando */
        unsigned int cantPartidas;

        /* Mapa de usuarios conectados. La clave corresponde al nombre del usuario */
        std::map<std::string,HiloManejadorCliente*> usuarios;

        /* Vector de escenarios disponibles para enviar a los usuarios */
        std::vector<std::string> escenarios;

        /* Ruta del archivo donde se almacenan los puntajes históricos de los usuarios */
        std::string rutaPuntajes;

        /* Mapa que almacena los puntajes de los usuarios. La clave es el nombre de usuario */
        std::map<std::string,unsigned int> puntajes;

        /* Vector que almacena todos los hilos de batallas en curso */
        std::vector<HiloBatalla*> desafios;

        /* Mutex que evita complicaciones en los hilos que leen o agregan mensajes recibidos
         * desde los clientes */
        Mutex mMensajesRecibidos;
        /* Vector de mensajes recibidos por los clientes */
        std::vector<Mensaje*> mensajesRecibidos;

        /* Carga el vector de escenarios en base a los que encuentra en el archivo de ruta
dada */
        void levantarEscenarios(const std::string& rutaEscenarios);

        /* Carga el vector de puntajes históricos en base a los datos que se encuentran en el
         * archivo dado */
        void levantarPuntajes();

        /* Guarda el vector de puntajes en el archivo de puntajes históricos */
        void grabarPuntajes();

        /* Analiza el mensaje dado y actúa en consecuencia llamando al método que corresponda
 */
        void analizarMensaje(Mensaje* mensaje);

        /* Quita al jugador de la lista de usuarios conectados y frena el hilo receptor de
         * datos */
        void desconectarJugador(std::string nombre);

        /* Inicia un juego individual para el usuario dado */
        void iniciarJuegoSolitario(std::string nombre);

        /* Envía mensaje de chat a todos los usuarios que no están jugando */
        void enviarMensajeChat(Mensaje* mensaje);

        /* Indica que el usuario deja de estar jugando y actualiza el puntaje */
        void finPartidoSolitario(Mensaje* mensaje);

        /* Envía una lista de usuarios conectados al jugador dado por parámetro */
        void enviarListaConectados(Jugador* jugador);
```

```
void enviarUsuarios(const std::string& nombre);

/* Envía notificación de invitación a desafío al usuario indicado en el texto del
mensaje */
void invitarDesafio(Mensaje* mensaje);

/* Notifica al usuario del texto del mensaje que su propuesta de juego ha sido
rechazada */
void notificarRechazo(Mensaje* mensaje);

/* Inicia el procedimiento de Juego Doble */
void aceptarDesafio(Mensaje* mensaje);

/* Devuelve un escenario al azar entre los disponibles */
bool pedirEscenario(std::string& nombre);

/* Elimina y cierra limpiamente los desafíos que se terminaron */
void mantenerDesafios();

/* Envía un mensaje a todos los usuarios que no están jugando */
void enviarMensajeATodos(const std::string& mensaje);

/* Devuelve una cadena con el ranking de posiciones cargadas */
void obtenerRanking(std::string& ranking, const std::string& jugador);

/* Obtiene el jugador que tiene menos puntos dentro del mapa dado */
void obtenerMenor(std::map<std::string, unsigned int>& lideres, std::string& menor,
unsigned int& ptosMenor);

/* Envía el ranking al usuario dado */
void enviarRanking(const std::string& nombre);

public:

/* Constructor. Recibe la ruta donde se almacenan los puntajes y donde se almacenan
* los escenarios */
AdministradorUsuarios(const std::string rutaPuntajes, const std::string
rutaEscenarios);

/* Destructor */
virtual ~AdministradorUsuarios();

/* Guarda en el parámetro el nombre de un escenario aleatorio que no haya sido
utilizado por el cliente.
* Devuelve false si no hay escenarios disponibles */
bool pedirEscenario(std::string& nombre, HiloManejadorCliente* cliente);

/* Incrementa el puntaje del usuario en la cantidad indicada */
bool actualizarPuntaje(std::string nombre, unsigned int puntos);

/* Agrega un jugador entre los que actualmente están conectados */
bool conectarJugador(const std::string nombre, Socket& socket);

/* Agrega un usuario nuevo a la base de datos de usuarios.
* Devuelve false si el nombre de usuario ya existe */
bool agregarUsuario(std::string nombre);

/* Analiza si el usuario existe o no */
bool usuarioExiste(std::string nombre);

/* Lógica principal del hilo */
void run();

};

#endif
```

```
#ifndef __HILO_BATALLA_H__
#define __HILO_BATALLA_H__

#include "server_HiloManejadorCliente.h"
#include "common_Thread.h"
#include "server_Mensaje.h"

class HiloBatalla: public Thread
{
    private:
        /* Guarda la referencia al la lista de mensajes recibidos del
         * administrador para devolverle el control luego */
        std::vector<Mensaje*>& mensajesRecibidosAdministrador;
        Mutex& mMensajesRecibidosAdministrador;

        /* Mensajes recibidos propios */
        std::vector<Mensaje*> mensajesRecibidos;

        /* Referencia a jugadores participantes */
        HiloManejadorCliente* jug1;
        HiloManejadorCliente* jug2;

        /* True si la partida está en curso. False si terminó */
        bool jugando;

        /* Nombre del escenario a utilizar */
        std::string escenario;

        /* Devuelve la referencia del jugador con el nombre dado */
        HiloManejadorCliente* obtenerJugador(const std::string& nombre);

        /* Devuelve la referencia del jugador contrario al del nombre dado */
        HiloManejadorCliente* obtenerRival(const std::string& nombre);

        /* Espero a ver qué resultó de cada una de las simulaciones de los jugadores
         * Devuelve true si debe terminarse la partida porque alguien ganó (por puntos o
abandono) */
        bool analizarResultado();

        //resultado:
        bool jugador1conectado;
        bool jugador2conectado;
        int puntosJug1;
        int puntosJug2;

    public:
        /* Constructor */
        HiloBatalla(HiloManejadorCliente* jug1, HiloManejadorCliente* jug2, std::string
escenario, Mutex& m, std::vector<Mensaje*>& mensajes);

        /* Preparación de la partida doble */
        bool iniciarPartida();

        /* Envía los archivos del escenario */
        bool enviarEscenario();

        /* True si el desafío sigue activo */
        bool activo();

        /* Lógica del hilo. Donde se juega la batalla */
        void run();

        /* Consulta de resultados */
        bool jug1Conectado();
        bool jug2Conectado();
        int jug1Puntos();
        int jug2Puntos();

        /* Obtención de las referencias de los jugadores */
        HiloManejadorCliente* obtenerJugador1();
```

```
HiloManejadorCliente* obtenerJugador2();  
};  
#endif
```

```
#ifndef __HILO_MANEJADOR_CLIENTE_H__
#define __HILO_MANEJADOR_CLIENTE_H__


#include "common_Socket.h"
#include "common_Thread.h"
#include "server_Mensaje.h"
#include <string>
#include <vector>

class HiloManejadorCliente: public Thread
{

private:
    Socket socket; //Socket que contacta al cliente.
    std::string nombre; //Nombre del cliente
    Mutex& mMensajesRecibidos; //Mutex para la lista de mensajes recibidos
    std::vector<Mensaje*>* mensajesRecibidos; //Lista de mensajes recibidos

    /* Vector de escenarios ya jugados por el usuario */
    std::vector<std::string> escenariosJugados;

    /* Indica si está en una partida o no */
    bool jugando;

    /* Convierte un nibble en un caracter hexadecimal */
    char ntohs(char n);

    /* Codifica en hexadecimal */
    void convertir(std::string& mensaje, char buf);

public:
    /* Constructor. Recibe el socket a través del cual se hará la conexión con el Cliente */
    HiloManejadorCliente(const std::string& nombre, const Socket& socket, Mutex& m,
    std::vector<Mensaje*>& mensajes);

    /* Destructor */
    ~HiloManejadorCliente();

    /* Va recibiendo los mensajes desde el cliente y los analiza */
    void run();

    /* Devuelve true si el cliente está jugando una partida */
    bool estaJugando();

    /* Modifica el atributo que indica si el Cliente está jugando o no */
    void setJugando(bool valor);

    /* Envía un mensaje de Chat al cliente */
    void enviarMensajeChat(Mensaje* mensaje);

    /* Envía un mensaje */
    void enviarMensaje(std::string mensaje);

    /* Redirecciona el flujo de almacenamiento de los mensajes recibidos */
    void setMensajesRecibidos(std::vector<Mensaje*>& mensajes);

    /* Devuelve la referencia a los mensajes recibidos */
    std::vector<Mensaje*>& getMensajesRecibidos();

    /* Envía un archivo codificado */
    void enviarArchivo(std::string ruta);

    /* Retorno el nombre del usuario */
    std::string& getNombre();

    /* Analiza si ya se jugó o no el escenario dado */
    bool escenarioJugado(const std::string& nombre);

    /* Agrega un escenario como ya jugado */
    void agregarEscenarioJugado(const std::string& nombre);
}
```

```
/* Vacía la lista de escenarios jugados */
void vaciarEscenariosJugados();

/* Shutdown del Socket */
void desconectar();

};

#endif
```

```
#ifndef __MENSAJE_H__
#define __MENSAJE_H__

#include <string>

class Mensaje
{
    private:
        std::string nombre; //Cliente que envió el mensaje
        char tipo; //Tipo de mensaje
        std::string texto; //Cuerpo del mensaje

    public:
        /* Constructor, recibe el mensaje completo y el nombre de quien lo envió */
        Mensaje(std::string mensaje, std::string nombre);

        /* Analiza si el mensaje es válido o no */
        bool esValido();

        /* Devuelve el nombre de quien lo envía */
        std::string& getNombre();

        /* Devuelve el tipo de mensaje */
        char getTipo();

        /* Devuelve el texto del mensaje */
        std::string& getTexto();
};

#endif
```

```
#ifndef __SERVER_H__
#define __SERVER_H__

#include "common_Thread.h"
#include "server_SocketServidor.h"
#include "common_Mutex.h"
#include <vector>
#include <iostream>
using namespace std;

class Server: public Thread
{
    private:
        int puerto; //Puerto del Servidor
        SocketServidor recepcionista; //Recibe nuevas conexiones

    public:
        /* Lógica principal del Server. Espera a que lleguen nuevos usuarios y se los agrega a la lista */
        void run();

        /* Inicializa el Servidor. Lo crea, hace el bind y listen */
        bool inicializar(const int puerto);
};

#endif
```

```
#ifndef __SOCKET_SERVIDOR_H__
#define __SOCKET_SERVIDOR_H__

#include "common_Socket.h"
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <iostream>

class SocketServidor: public Socket
{
public:
    /* Asigna el puerto al que se asocia el socket.
     * Se utiliza para los servidores y el puerto pasado
     * es el puerto al que se conectarán los clientes
     * Devuelve true si fue una operación exitosa. */
    bool bind(const int puerto);

    /* Para un servidor.
     * Establece la cantidad de conexiones en espera que
     * puede tener hasta hacer un accept.
     * Empieza a "escuchar" conexiones entrantes.
     * Devuelve true si fue una operación exitosa. */
    bool listen() const;

    /* Acepta una nueva conexión que se conecta a nosotros.
     * Carga en nuevo el Socket que surge de este proceso y a partir
     * del cual se empezarán a comunicar.
     * Devuelve true si fue una operación exitosa. */
    bool accept(Socket& nuevo);

};

#endif
```

```
#ifndef __STRING_ENTERO_H__
#define __STRING_ENTERO_H__

#include <string>

class StringEnter
{
public:
    /* Convierte el entero dado a un string */
    static void enteroAString(std::string& cadena, ulong entero);
};

#endif
```

```
#include "common_Mutex.h"

Mutex::Mutex()
{
    pthread_mutex_init(&mut, NULL);
}

void Mutex::lock()
{
    pthread_mutex_lock(&mut);
}

void Mutex::unlock()
{
    pthread_mutex_unlock(&mut);
}
```

```
#include "common_Socket.h"
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <iostream>
#include <iostream>
using namespace std;

// flag externo que indica si debe mostrar o no los paquetes
extern bool modoDebug;

Socket::Socket()
{
    //Aún no está creado
    id = SOCKET_INVALIDO;

    //Inicializa dirección y buffer de remanentes
    memset(&(dir.sin_zero), '\0', 8);
    memset(historial, 0, TAM_BUFFER + 1 );
    validosHistorial = 0;
}

Socket::~Socket()
{
    if (id != SOCKET_INVALIDO)
        ::close(id); //Cierra Socket
}

bool Socket::create()
{
    //Crea Socket
    id = socket(AF_INET, SOCK_STREAM, 0);

    //Analiza resultado
    if (id == SOCKET_INVALIDO)
        return false;
    return true;
}

bool Socket::send(const string& dato) const
{
    unsigned int enviados = 0;
    int retorno;
    string aux = dato;

    /* Envía hasta que la cantidad de enviados sea el tamaño del dato más el \0
     * que indica fin de mensaje. */
    while (enviados != (dato.size() + 1))
    {
        retorno = ::send (id, aux.c_str(), dato.size() + 1 - enviados, MSG_NOSIGNAL);
        if (retorno == -1)
            return false;
        enviados += retorno; //Suma los enviados.
        if (enviados != (dato.size() + 1))
            aux = dato.substr(enviados); //Carga en aux los datos que faltan enviar
    }

    if (modoDebug)
        cerr << "Enviado: " << dato << endl;
}

return true;
}

bool Socket::recv(string& dato)
{
    char buffer [TAM_BUFFER + 1];
```

```
bool listo = false;
dato = "";
int retorno;
unsigned int recibidos;

//Analiza si hay elementos remanentes que son parte del mensaje actual.
if (validosHistorial != 0)
{
    dato = historial; //Toma hasta el primer \0
    if (dato.size() != validosHistorial) //Si el dato no tomó todos los remanentes es
porque encontró un \0
    {
        listo = true;
        validosHistorial -= (dato.size()+1); //Resto el tamaño del mensaje mas el \0
    }
else
    validosHistorial = 0;

//Coloca en historial los nuevos remanentes.
memcpy(buffer,historial+dato.size()+1,validosHistorial);
memset(historial, 0, TAM_BUFFER + 1 );
memcpy(historial,buffer,validosHistorial);

}

//Inicializa el buffer
memset (buffer,0,TAM_BUFFER + 1 );

//Recibe hasta encontrar el \0
while (!listo)
{
    retorno = ::recv(id,buffer,TAM_BUFFER, 0 );
    if ((retorno == -1) || (retorno == 0))
        return false;
    recibidos = retorno;
    if (strlen(buffer) < recibidos)
    {
        listo = true;
        if (strlen(buffer) < recibidos - 1)
            //Tengo que copiar a los remanentes
            validosHistorial = recibidos-(strlen(buffer)+1);
            memcpy(historial,buffer+strlen(buffer)+1,validosHistorial);
    }
    dato += buffer; //Concatena lo leído al dato.
    memset(buffer, 0, TAM_BUFFER + 1 );
}

if (modoDebug)
    cerr << "Recibido: " << dato << endl;

return true;
}

void Socket::setId(const int id)
{
    this->id = id;
}

int Socket::getId() const
{
    return id;
}

sockaddr_in& Socket::getDir()
{
    return dir;
}

void Socket::close()
{
```

```
if (id == SOCKET_INVALIDO)
    return;
::close(id);
id = SOCKET_INVALIDO;
}

void Socket::shutdown(int modo)
{
    if (id == SOCKET_INVALIDO)
        return;
    ::shutdown(id,modo);
    id = SOCKET_INVALIDO;
}
```

```
#include "common_SocketCliente.h"
#include <iostream>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <string>
using namespace std;

bool SocketCliente::connect(const string direccion, const int puerto)
{
    int id = getId();
    sockaddr_in& dir = getDir();

    if (id == SOCKET_INVALIDO)
    {
        if (!this->create())
            return false;
        id = this->getId();
    }

    //Prepara la conexión
    struct hostent *he;
    he = gethostbyname(direccion.c_str());

    if (he == NULL)
        return false;

    dir.sin_family = AF_INET;
    dir.sin_port = htons (puerto);
    dir.sin_addr = *((struct in_addr *)he->h_addr);

    //Intenta conectarse.
    if (::connect(id,(sockaddr*)&dir,sizeof(dir)) == -1)
        return false;

    return true;
}
```

```
#include "common_Thread.h"

#include <signal.h>
#include <unistd.h>

Thread::Thread()
{
    vivo = true;
}

void Thread::start()
{
    corriendo = true;
    pthread_create(&hilo,NULL,static_run,this);
}

Thread::~Thread()
{
    if (corriendo)
    {
        vivo = false; //Indica al hilo que debe suicidarse.
        this->join();
    }
}

void Thread::join() const
{
    pthread_join(hilo,NULL);
}

bool Thread::estaVivo()
{
    bool retorno;
    bloquearEstado();
    retorno = vivo;
    desbloquearEstado();
    return retorno;
}

void* Thread::static_run (void* p)
{
    Thread* pthis = (Thread*) p;
    pthis->run();
    return NULL;
}

void Thread::terminar()
{
    bloquearEstado();
    vivo = false;
    desbloquearEstado();
}

bool Thread::estaCorriendo()
{
    bool retorno;
    bloquearEstado();
    retorno = corriendo;
    desbloquearEstado();
    return retorno;
}

void Thread::bloquearEstado()
```

```
mHilo.lock();  
}  
  
void Thread::desbloquearEstado()  
{  
    mHilo.unlock();  
}  
  
void Thread::setCorriendo(bool valor)  
{  
    corriendo = valor;  
}
```

```
#include "server_AdministradorUsuarios.h"
#include "StringEntero.h"
#include <map>
#include <cstdlib>
#include <fstream>
#include <iostream>
using namespace std;

#define CANT_LIDERES 5

void AdministradorUsuarios::levantarEscenarios(const std::string& rutaEscenarios)
{
    ifstream arch(rutaEscenarios.c_str());
    string linea;

    //Cada nombre de escenario es una linea
    while (getline(arch,linea,'`n'))
        escenarios.push_back(linea);
    arch.close();
}

void AdministradorUsuarios::levantarPuntajes()
{
    string nombre;
    string puntos;
    ifstream arch(rutaPuntajes.c_str());
    while (!arch.eof())
    {
        // Lee el nombre seguido del puntaje
        arch >> nombre;
        if (!arch.eof())
        {
            arch >> puntos;
            puntajes.insert(make_pair(nombre,atoi(puntos.c_str())));
        }
    }
    arch.close();
}

AdministradorUsuarios::AdministradorUsuarios(const std::string rutaPuntajes, const std::string rutaEscenarios)
{
    this->rutaPuntajes = rutaPuntajes;
    levantarPuntajes();
    levantarEscenarios(rutaEscenarios);
    cantPartidas = 0;
}

void AdministradorUsuarios::grabarPuntajes()
{
    ofstream arch(rutaPuntajes.c_str());
    map<string,unsigned int>::iterator it = puntajes.begin();
    // Graba un nombre de usuario seguido de su correspondiente puntaje
    while (it != puntajes.end())
    {
        arch << it->first << " " << it->second << " ";
        it++;
    }
    arch.close();
}

AdministradorUsuarios::~AdministradorUsuarios()
{
    grabarPuntajes();

    //elimina los hilos batallas
    vector<HiloBatalla*>::iterator itDes = desafios.begin();
    HiloBatalla* actual;
    while (itDes != desafios.end())
    {
```

```
        actual = *itDes;
        actual->terminar();
        actual->join();
        delete actual;
        itDes++;
    }

//elimina los hilos manejadores de clientes
map<string,HiloManejadorCliente*>::iterator it = usuarios.begin();
while (it != usuarios.end())
{
    it->second->desconectar();
    it->second->terminar();
    it->second->join();
    delete it->second;
    it++;
}

//elimina los mensajes remanentes
Mensaje* mensaje;
mMensajesRecibidos.lock();
while (!mensajesRecibidos.empty())
{
    mensaje = mensajesRecibidos[0];
    mensajesRecibidos.erase(mensajesRecibidos.begin());
    delete mensaje;
}
mMensajesRecibidos.unlock();

bool AdministradorUsuarios::pedirEscenario(std::string& nombre, HiloManejadorCliente* cliente)
{
    if (escenarios.size() == 0)
        return false; // no hay escenarios

    vector<string> disponibles;

    //Obtiene los escenarios que todavía no jugó este usuario
    for (unsigned int i = 0 ; i < escenarios.size() ; i++)
        if (!cliente->escenarioJugado(escenarios[i]))
            disponibles.push_back(escenarios[i]);

    if (disponibles.size() == 0)
        return false; // ya jugó todos los escenarios

    //Elije un nombre random
    nombre = disponibles[(rand() % disponibles.size())];
    cliente->agregarEscenarioJugado(nombre);
    nombre = "esc/" + nombre;

    return true;
}

bool AdministradorUsuarios::pedirEscenario(std::string& nombre)
{
    if (escenarios.size() == 0)
        return false; // no hay escenarios

    //Elije un nombre random
    nombre = escenarios[(rand() % escenarios.size())];
    nombre = "esc/" + nombre;

    return true;
}

bool AdministradorUsuarios::actualizarPuntaje(std::string nombre, unsigned int puntos)
{
    bool retorno;
    bloquearEstado();
```

```
// busca al usuario
map<string,unsigned int>::iterator it = puntajes.find(nombre);
if (it != puntajes.end())
{
    //suma puntos
    it->second += puntos;
    retorno = true;
}
else
    retorno = false; //No existe el usuario
desbloquearEstado();
return retorno;
}

bool AdministradorUsuarios::usuarioExiste(std::string nombre)
{
    bool retorno;
    bloquearEstado();
    // hace búsqueda del usuario y analiza su existencia
    map<string,unsigned int>::iterator it = puntajes.find(nombre);
    if (it != puntajes.end())
        retorno = true;
    else
        retorno = false; //No existe el usuario
    desbloquearEstado();
    return retorno;
}

bool AdministradorUsuarios::conectarJugador(const std::string nombre, Socket& socket)
{
    bool retorno;
    HiloManejadorCliente* manejador;

    bloquearEstado();

    //Busca si existe en nombre
    map<string,HiloManejadorCliente*>::iterator it = usuarios.find(nombre);

    if (it == usuarios.end())
    {
        // informa al resto que se conectó el usuario
        enviarMensajeATodos("C " + nombre + " se ha conectado");

        // crea el nuevo manejador del cliente
        manejador = new HiloManejadorCliente
    (nombre,socket,mMensajesRecibidos,mensajesRecibidos);
        usuarios.insert(make_pair(nombre,manejador));

        //Hace correr al hilo
        manejador->start();
        retorno = true;
    }
    else
        retorno = false; //No puede existir el usuario
    desbloquearEstado();

    return retorno;
}

bool AdministradorUsuarios::agregarUsuario(std::string nombre)
{
    bool retorno;
    bloquearEstado();

    //Busca si existe en nombre
    map<string,unsigned int>::iterator it = puntajes.find(nombre);
```

```
if (it == puntajes.end())
{
    //agrega el usuario
    puntajes.insert(make_pair(nombre,0));
    retorno = true;
}
else
    retorno = false; //No puede existir el usuario previamente

desbloquearEstado();

}

return retorno;
}

void AdministradorUsuarios::desconectarJugador(std::string nombre)
{

bloquearEstado();
map<string,HiloManejadorCliente*>::iterator it = usuarios.find(nombre); //Busca al usuario
if (it != usuarios.end())
{
    // mensaje que desbloquea el socket del cliente
    it->second->enviarMensaje("D");

    // termina y elimina el hilo que maneja al cliente
    it->second->join();
    delete it->second;
    usuarios.erase(it);
}
//envía mensaje de desconexión al resto de los usuarios
enviarMensajeATodos("C " + nombre + " se ha desconectado");
desbloquearEstado();
}

void AdministradorUsuarios::iniciarJuegoSolitario(std::string nombre)
{
    string escenario,fondo,posiciones;

    bloquearEstado();
    map<string,HiloManejadorCliente*>::iterator it = usuarios.find(nombre);
    if (it != usuarios.end())
    {
        if (cantPartidas == MAX_PARTIDAS)
            it->second->enviarMensaje("0"); //ya está lleno el cupo de partidas
        else
        {
            // pide escenario que no se haya jugado
            if (pedirEscenario(escenario,it->second))
            {
                // envía los dos archivos correspondientes al escenario
                fondo = escenario + ".jpg";
                posiciones = escenario + ".xml";
                it->second->enviarArchivo(fondo);
                it->second->enviarArchivo(posiciones);
                it->second->setJugando(true);
                cantPartidas++;

            }
            else
            {
                it->second->enviarMensaje("T"); //Termino todos los niveles
                it->second->vaciarEscenariosJugados();
            }
        }
    }
    desbloquearEstado();
}
```

```
void AdministradorUsuarios::enviarMensajeChat(Mensaje* mensaje)
{
    bloquearEstado();
    map<string,HiloManejadorCliente*>::iterator it = usuarios.begin();
    while (it != usuarios.end())
    {
        // envía el mensaje a todos los que no están jugando
        if (!(it->second->estaJugando()))
            it->second->enviarMensajeChat(mensaje);
        it++;
    }
    desbloquearEstado();
}

void AdministradorUsuarios::enviarMensajeATodos(const std::string& mensaje)
{
    map<string,HiloManejadorCliente*>::iterator it = usuarios.begin();
    while (it != usuarios.end())
    {
        // envía el mensaje a todos los usuarios que no están jugando
        if (!(it->second->estaJugando()))
            it->second->enviarMensaje(mensaje);
        it++;
    }
}

void AdministradorUsuarios::finPartidoSolitario(Mensaje* mensaje)
{
    bloquearEstado();

    map<string,HiloManejadorCliente*>::iterator it = usuarios.find(mensaje->getNombre());

    // indica que no está jugando
    if (it != usuarios.end())
        it->second->setJugando(false);

    cantPartidas--;
    desbloquearEstado();
    actualizarPuntaje(mensaje->getNombre(),atoi((mensaje->getTexto()).c_str()));
}

void AdministradorUsuarios::enviarUsuarios(const std::string& nombre)
{
    string mensaje = "U";
    HiloManejadorCliente* hilo = NULL;

    bloquearEstado();

    map<string,HiloManejadorCliente*>::iterator it = usuarios.begin();

    // concatena los nombres de usuarios distintos al dado con un \n que separa
    while (it != usuarios.end())
    {
        if (nombre.compare(it->first) != 0)
            mensaje += it->first + "\n";
        else
            hilo = it->second;
        it++;
    }

    // envía el mensaje
    if (hilo != NULL)
        hilo->enviarMensaje(mensaje);
```

```
desbloquearEstado();  
}  
  
void AdministradorUsuarios::invitarDesafio(Mensaje* mensaje)  
{  
    string propuesta = "B";  
    propuesta += mensaje->getNombre();  
  
    bloquearEstado();  
  
    map<string,HiloManejadorCliente*>::iterator it = usuarios.find(mensaje->getTexto());  
  
    if ((it != usuarios.end()) && (!it->second->estaJugando()))  
        it->second->enviarMensaje(propuesta); // envía propuesta  
    else  
    {  
        it = usuarios.find(mensaje->getNombre());  
        propuesta = "R";  
        propuesta += mensaje->getTexto();  
        if (it == usuarios.end())  
            propuesta += "\nNo está conectado el usuario";  
        else  
            propuesta += "\nEl usuario ya está jugando";  
        it->second->enviarMensaje(propuesta);  
    }  
  
    desbloquearEstado();  
}  
  
void AdministradorUsuarios::notificarRechazo(Mensaje* mensaje)  
{  
    string notificacion = "R";  
    notificacion += mensaje->getNombre();  
    notificacion += "\nHa rechazado su propuesta.";  
  
    bloquearEstado();  
    map<string,HiloManejadorCliente*>::iterator it = usuarios.find(mensaje->getTexto());  
  
    if (it != usuarios.end())  
        //Envío notificación de rechazo de desafío  
        it->second->enviarMensaje(notificacion);  
  
    desbloquearEstado();  
}  
  
void AdministradorUsuarios::aceptarDesafio(Mensaje* mensaje)  
{  
    string notificacion;  
  
    //Tengo que analizar si el desafiante no empezó a jugar mientras el desafiado aceptaba  
    bloquearEstado();  
  
    map<string,HiloManejadorCliente*>::iterator it = usuarios.find(mensaje->getTexto());  
  
    //Primero ver si está conectado todavía  
    if (it == usuarios.end())  
    {  
        notificacion = "R";  
        notificacion += mensaje->getTexto();  
        notificacion += "\nYa no está conectado.";  
    }  
    //Analiza si no empezó a jugar otra partida  
    else if (it->second->estaJugando())  
    {  
        notificacion = "R";  
        notificacion += mensaje->getTexto();  
        notificacion += "\nEstá jugando otra partida.";  
    }  
}
```

```
    }
    else
    {
        HiloManejadorCliente* j1;
        HiloManejadorCliente* j2;
        HiloBatalla* nuevoHilo;
        string escenario;

        //Arranca todo el proceso de una partida multijugador

        //Envío mensaje de desafío.
        j1 = it->second;
        it = usuarios.find(mensaje->getNombre());
        j2 = it->second;

        if (cantPartidas == MAX_PARTIDAS)
        {
            // cupo de partidas ocupado
            j1->enviarMensaje("0");
            j2->enviarMensaje("0");
        }
        else
        {
            pedirEscenario(escenario);

            // indica inicio de partida
            j1->enviarMensaje("Z"+mensaje->getNombre());
            j2->enviarMensaje("Z"+mensaje->getTexto());

            //crea hilo batalla
            nuevoHilo = new HiloBatalla
(j1,j2,escenario,mMensajesRecibidos,mMensajesRecibidos);
            desafios.push_back(nuevoHilo);
            nuevoHilo->iniciarPartida(); //inicializa la batalla
            nuevoHilo->start();
            cantPartidas++;
        }
    }

    desbloquearEstado();

}

void AdministradorUsuarios::analizarMensaje(Mensaje* mensaje)
{
    // actúa en función del tipo de mensaje
    char tipo = mensaje->getTipo();
    switch (tipo)
    {
        case 'D':
            desconectarJugador(mensaje->getNombre());
            break;
        case 'J':
            iniciarJuegoSolitario(mensaje->getNombre());
            break;
        case 'C':
            enviarMensajeChat(mensaje);
            break;
        case 'P':
            finPartidoSolitario(mensaje);
            break;
        case 'U':
            enviarUsuarios(mensaje->getNombre());
            break;
        case 'B':
            invitarDesafio(mensaje);
            break;
        case 'R':
            notificarRechazo(mensaje);
            break;
    }
}
```

```
        case 'A':
            aceptarDesafio(mensaje);
            break;
        case 'G':
            enviarRanking(mensaje->getNombre());
            break;
    }
}

void AdministradorUsuarios::enviarRanking(const std::string& nombre)
{
    string ranking;

    //busca la tabla de posiciones
    obtenerRanking(ranking,nombre);

    bloquearEstado();
    map<string,HiloManejadorCliente*>::iterator it = usuarios.find(nombre);
    if (it != usuarios.end())
    {
        it->second->enviarMensaje(ranking);
    }
    desbloquearEstado();
}

void AdministradorUsuarios::mantenerDesafios()
{
    vector<HiloBatalla*>::iterator it = desafios.begin();
    HiloBatalla* actual;
    while (it != desafios.end())
    {
        actual = *it;
        if (!actual->activo())
        {
            //Analizar resultado
            if (actual->jug1Puntos() > 0)
                actualizarPuntaje(actual->obtenerJugador1()->getNombre(),actual-
>jug1Puntos());
            if (actual->jug2Puntos() > 0)
                actualizarPuntaje(actual->obtenerJugador2()->getNombre(),actual-
>jug2Puntos());

            //Analiza si se desconecto alguien
            if (!actual->jug1Conectado())
                desconectarJugador(actual->obtenerJugador1()->getNombre());
            if (!actual->jug2Conectado())
                desconectarJugador(actual->obtenerJugador2()->getNombre());

            //Limpia lo que quedó del desafío
            actual->join();
            delete actual;
            desafios.erase(it);
            it = desafios.begin();
        }
        else
            it++;
    }
}

void AdministradorUsuarios::obtenerRanking(std::string& ranking, const std::string& jugador)
{
    ranking = "";
    unsigned int posJugador = 1;
    unsigned int puntosJugador;
    string menor;
    unsigned int ptosMenor;

    map<string,unsigned int> lideres;
```

```
bloquearEstado();

// busca los puntos del jugador que pide la tabla
map<string,unsigned int>::iterator it = puntajes.find(jugador);
if (it == puntajes.end())
{
    desbloquearEstado();
    return;
}
puntosJugador = it -> second;

it = puntajes.begin();

while (it != puntajes.end())
{
    if (jugador.compare(it -> first) != 0)
        if (it->second > puntosJugador)
            posJugador++; // cae una posición
    if (lideres.size() < CANT_LIDERES)
        lideres.insert(make_pair(it->first,it->second));
    else
    {
        obtenerMenor(lideres,menor,ptosMenor);
        for (unsigned int i = 0 ; i < CANT_LIDERES ; i++)
        {
            if (ptosMenor < it->second)
            {
                //agrega a los líderes
                lideres.erase(menor);
                lideres.insert(make_pair(it->first,it->second));
            }
        }
        it++;
    }
}

//hasta acá tengo el mapa con los líderes hay que ordenarlo.
unsigned int i = lideres.size();
string nro;

//imprimo posicion del que pido el ranking
StringEntero::enteroAString(nro,puntosJugador);
ranking = " " + nro; //puntos
StringEntero::enteroAString(nro,posJugador);
ranking = "Tu posición: \n" + nro + ". " + jugador + ranking;
ranking = "\n\n" + ranking;

while (!lideres.empty())
{
    obtenerMenor(lideres,menor,ptosMenor);

    StringEntero::enteroAString(nro,ptosMenor);
    ranking = menor + " " + nro + ranking;

    StringEntero::enteroAString(nro,i);
    ranking = "\n" + nro + ". " + ranking;

    lideres.erase(menor);

    i--;
}

ranking = "GTabla de posiciones\n" + ranking;
desbloquearEstado();

}

void AdministradorUsuarios::obtenerMenor(map<string,unsigned int>& lideres, std::string& menor,
unsigned int& ptosMenor)
```

```
{  
    map<string,unsigned int>::iterator it = lideres.begin();  
    ptosMenor = it-> second;  
    menor = it->first;  
    it++;  
    while (it != lideres.end())  
    {  
        if (it->second < ptosMenor)  
        {  
            menor = it->first;  
            ptosMenor = it->second;  
        }  
        it++;  
    }  
  
}  
  
void AdministradorUsuarios::run()  
{  
    Mensaje* mensaje;  
    while (estaVivo())  
    {  
        mMensajesRecibidos.lock();  
        //saca un mensaje de la lista  
        if (!mensajesRecibidos.empty())  
        {  
            mensaje = mensajesRecibidos[0];  
            mensajesRecibidos.erase(mensajesRecibidos.begin());  
        }  
        else  
            mensaje = NULL;  
        mMensajesRecibidos.unlock();  
  
        if (mensaje != NULL)  
        {  
            analizarMensaje(mensaje); //Analiza el mensaje y actúa en consecuencia  
            delete mensaje;  
        }  
  
        mantenerDesafios();  
  
        sleep(0);  
    }  
    setCorriendo(false);  
}
```

```
#include "server_HiloBatalla.h"
#include <iostream>
using namespace std;

HiloBatalla::HiloBatalla(HiloManejadorCliente* jug1, HiloManejadorCliente* jug2, std::string
escenario, Mutex& m, std::vector<Mensaje*>& mensajes) : mensajesRecibidosAdministrador (mensajes),
mMensajesRecibidosAdministrador(m)
{
    //inicializa atributos

    this->jug1 = jug1;
    this->jug2 = jug2;
    this->escenario = escenario;
    this->jugando = true;

    jugador1conectado = true;
    jugador2conectado = true;
    puntosJug1 = -1;
    puntosJug2 = -1;
}

bool HiloBatalla::iniciarPartida()
{
    bool listo1 = false;
    bool listo2 = false;

    jug1->setJugando(true);
    jug2->setJugando(true);

    //Redirecciona el flujo de los mensajes
    jug1->setMensajesRecibidos(mensajesRecibidos);
    jug2->setMensajesRecibidos(mensajesRecibidos);

    //Envía los archivos correspondientes al escenario elegido
    enviarEscenario();

    //Espera a que los dos estén listos pues recibieron los archivos.
    Mensaje* mensaje;
    while ((!listo1) || (!listo2))
    {

        mMensajesRecibidosAdministrador.lock();

        if (!mensajesRecibidos.empty())
        {
            mensaje = mensajesRecibidos[0];
            mensajesRecibidos.erase(mensajesRecibidos.begin());

            //Pregunta por tipo listo
            if (mensaje->getTipo() == 'L')
            {

                //Cuál está listo?
                if ((mensaje->getNombre()).compare(jug1->getNombre()) == 0)
                    listo1 = true;
                else
                    listo2 = true;
            }

            delete mensaje;
        }
        mMensajesRecibidosAdministrador.unlock();
        sleep(0);
    }
}
```

```
//Envía la señal de que pueden jugar
jug1->enviarMensaje("J");
jug2->enviarMensaje("J");

return true;
}

bool HiloBatalla::enviarEscenario()
{
    //Envía fondo
    jug1->enviarArchivo(escenario+".jpg");
    jug2->enviarArchivo(escenario+".jpg");

    //Envía posiciones
    jug1->enviarArchivo(escenario+".xml");
    jug2->enviarArchivo(escenario+".xml");

    return true;
}

bool HiloBatalla::activo()
{
    bool ret;
    bloquearEstado();
    ret = jugando;
    desbloquearEstado();
    return ret;
}

void HiloBatalla::run()
{
    bool finPartida = false;
    bool listo1 = false;
    bool listo2 = false;

    Mensaje* mensaje;

    //Ejecuta la partida
    while (!finPartida) && (estaVivo())
    {

        mMensajesRecibidosAdministrador.lock();

        if (!mensajesRecibidos.empty())
        {
            mensaje = mensajesRecibidos[0];
            mensajesRecibidos.erase(mensajesRecibidos.begin());

            switch (mensaje->getTipo())
            {
                case 'C':
                    //Mensaje de chat. Debe enviarse a los dos con distintos
encabezados
                    obtenerJugador(mensaje->getNombre())->enviarMensaje("Ctu: " +
mensaje->getTexto());
                    obtenerRival(mensaje->getNombre())->enviarMensaje("Cel: " +
mensaje->getTexto());
                    break;
                case 'M':
                    //Envia un movimiento.
                    obtenerRival(mensaje->getNombre())->enviarMensaje("M" +
mensaje->getTexto());
                    break;
                case 'L':
                    //Un jugador listo para simular
                    if (listo1)
                        listo2 = true;
                    else
                    {
                        listo1 = true;
                        obtenerJugador(mensaje->getNombre())->enviarMensaje
("X");
                    }
            }
        }
    }
}
```

```
        }
        break;
    case 'D':
        //se desconectó uno de los jugadores

        //informar al otro
        obtenerRival(mensaje->getNombre())->enviarMensaje("H");
        //ajustar el resultado
        if ((mensaje->getNombre()).compare(jug1->getNombre()) == 0)
            jugador1conectado = false;
        else
            jugador2conectado = false;
        finPartida = true;
        break;
    case 'H':
        //abandonó uno de los jugadores
        //informar al otro
        obtenerRival(mensaje->getNombre())->enviarMensaje("H");
        finPartida = true;
        break;
    }
}

delete mensaje;
}

mMensajesRecibidosAdministrador.unlock();

if ((listo1) && (listo2))
{
    //envía simulación
    jug1->enviarMensaje("S");
    jug2->enviarMensaje("S");

    //espero respuesta de cada uno
    if (analizarResultado())
        finPartida = true; //alguien gano
    else
    {
        listo1 = false;
        listo2 = false;
    }
}

sleep(0);

}

if ((estaVivo()) && (listo1) && (listo2))
{
    string leyendaResultado = "";

    // mando resultado
    if (puntosJug1 > puntosJug2)
        leyendaResultado = jug1->getNombre() + " gana la partida";
    else
    {
        if (puntosJug1 < puntosJug2)
            leyendaResultado = jug2->getNombre() + " gana la partida";
        else
            leyendaResultado = "Ha empatado la partida";
    }

    leyendaResultado = "I" + leyendaResultado;
    jug1->enviarMensaje(leyendaResultado);
    jug2->enviarMensaje(leyendaResultado);
}

//devuelve el control al administrador
jug1->setMensajesRecibidos(mensajesRecibidosAdministrador);
jug2->setMensajesRecibidos(mensajesRecibidosAdministrador);
```

```
jug1->setJugando(false);
jug2->setJugando(false);

bloquearEstado();
jugando = false;
desbloquearEstado();
}

bool HiloBatalla::analizarResultado()
{
    Mensaje* mensaje;
    bool obtuveRespuesta1 = false;
    bool obtuveRespuesta2 = false;
    bool retorno = false;

    while ((!obtuveRespuesta1) || (!obtuveRespuesta2))
    {

        mMensajesRecibidosAdministrador.lock();

        if (!mensajesRecibidos.empty())
        {
            mensaje = mensajesRecibidos[0];
            mensajesRecibidos.erase(mensajesRecibidos.begin());

            switch (mensaje->getTipo())
            {
                case 'D':
                    //informar al otro
                    obtenerRival(mensaje->getNombre())->enviarMensaje("H");
                    //ajustar el resultado
                    if ((mensaje->getNombre()).compare(jug1->getNombre()) == 0)
                        jugador1conectado = false;
                    else
                        jugador2conectado = false;
                    retorno = true;
                    this->terminar();
                    break;

                case 'P':
                    //recibe puntaje
                    if ((mensaje->getNombre()).compare(jug1->getNombre()) == 0)
                    {
                        puntosJug1 = atoi((mensaje->getTexto()).c_str());
                        obtuveRespuesta1 = true;
                    }
                    else
                    {
                        puntosJug2 = atoi((mensaje->getTexto()).c_str());
                        obtuveRespuesta2 = true;
                    }
                    retorno = true;
                    obtuveRespuesta1 = true;
                    obtuveRespuesta2 = true;
                    break;

                case 'N':
                    // no termino el mapa
                    if ((mensaje->getNombre()).compare(jug1->getNombre()) == 0)
                        obtuveRespuesta1 = true;
                    else
                        obtuveRespuesta2 = true;
                    break;
            }

            delete mensaje;
        }

        mMensajesRecibidosAdministrador.unlock();
    }
}
```

```
    sleep(0);

}

return retorno;

}

HiloManejadorCliente* HiloBatalla::obtenerJugador(const std::string& nombre)
{
    if (nombre.compare(jug1->getNombre()) == 0)
        return jug1;
    else
        return jug2;
}

HiloManejadorCliente* HiloBatalla::obtenerRival(const std::string& nombre)
{
    if (nombre.compare(jug1->getNombre()) == 0)
        return jug2;
    else
        return jug1;
}

bool HiloBatalla::jug1Conectado()
{
    return jugador1conectado;
}

bool HiloBatalla::jug2Conectado()
{
    return jugador2conectado;
}

int HiloBatalla::jug1Puntos()
{
    return puntosJug1;
}

int HiloBatalla::jug2Puntos()
{
    return puntosJug2;
}

HiloManejadorCliente* HiloBatalla::obtenerJugador1()
{
    return jug1;
}

HiloManejadorCliente* HiloBatalla::obtenerJugador2()
{
    return jug2;
}
```

```
#include "server_HiloManejadorCliente.h"
#include <fstream>
using namespace std;

HiloManejadorCliente::HiloManejadorCliente(const string& nombre, const Socket& socket,Mutex& m,
vector<Mensaje*>& mensajes): mMensajesRecibidos(m) , mensajesRecibidos(&mensajes)
{
    this->socket = socket;
    this->nombre = nombre;
    this->jugando = false;
}

HiloManejadorCliente::~HiloManejadorCliente()
{
    socket.close();
}

void HiloManejadorCliente::desconectar()
{
    socket.shutdown(0);
}

void HiloManejadorCliente::run()
{
    string cadMensaje;
    Mensaje* mensaje;
    bool pideFin = false;

    while (!pideFin)
    {
        if (!socket.recv(cadMensaje))
        {
            //Desapareció el Cliente
            cadMensaje = "D";
            pideFin = true;
        }
        else if (cadMensaje[0] == 'D')
            pideFin = true;

        mensaje = new Mensaje(cadMensaje,nombre);

        //Coloca en la lista el mensaje junto a su nombre.
        mMensajesRecibidos.lock();
        mensajesRecibidos->push_back(mensaje);
        mMensajesRecibidos.unlock();
        sleep(0);
    }
    setCorriendo(false);
}

bool HiloManejadorCliente::estaJugando()
{
    return jugando;
}

void HiloManejadorCliente::setJugando(bool valor)
{
    this->jugando = valor;
}

void HiloManejadorCliente::enviarMensajeChat(Mensaje* mensaje)
{
    socket.send("C" + mensaje->getNombre() + " dice: " + mensaje->getTexto());
}

void HiloManejadorCliente::enviarMensaje(string mensaje)
{
    socket.send(mensaje);
}
```

```
void HiloManejadorCliente::setMensajesRecibidos(std::vector<Mensaje*>& mensajes)
{
    mMensajesRecibidos.lock();
    mensajesRecibidos = &mensajes;
    mMensajesRecibidos.unlock();
}

std::vector<Mensaje*>& HiloManejadorCliente::getMensajesRecibidos()
{
    return *mensajesRecibidos;
}

void HiloManejadorCliente::enviarArchivo(string ruta)
{
    ifstream arch(ruta.c_str(),ios::binary);
    string mensaje = "A";
    char buf;

    while (!arch.eof())
    {
        arch.get(buf);
        if (!arch.eof())
            convertir(mensaje,buf);
    }

    socket.send(mensaje);

    arch.close();
}

char HiloManejadorCliente::ntoh(char n)
{
    switch (n)
    {
        case 0: return '0';
        case 1: return '1';
        case 2: return '2';
        case 3: return '3';
        case 4: return '4';
        case 5: return '5';
        case 6: return '6';
        case 7: return '7';
        case 8: return '8';
        case 9: return '9';
        case 10: return 'a';
        case 11: return 'b';
        case 12: return 'c';
        case 13: return 'd';
        case 14: return 'e';
        case 15: return 'f';
    }
    return -1;
}

void HiloManejadorCliente::convertir(string& mensaje, char buf)
{
    // codifica el archivo

    char nibble;
    char aux[3];
    aux[2] = '\0';

    nibble = buf >> 4;
    nibble = nibble & 15;

    aux[0] = ntohs(nibble);

    nibble = buf & 15;
    aux[1] = ntohs(nibble);

    mensaje += aux;
}
```

```
string& HiloManejadorCliente::getNombre()
{
    return nombre;
}

bool HiloManejadorCliente::escenarioJugado(const std::string& nombre)
{
    bool retorno = false;
    bloquearEstado();

    // recorre los escenarios jugados a ver si encuentra el dado
    for (unsigned int i = 0 ; i < escenariosJugados.size() ; i++)
        if (nombre.compare(escenariosJugados[i]) == 0)
    {
        retorno = true;
        break;
    }

    desbloquearEstado();
    return retorno;
}

void HiloManejadorCliente::agregarEscenarioJugado(const std::string& nombre)
{
    bloquearEstado();
    escenariosJugados.push_back(nombre);
    desbloquearEstado();
}

void HiloManejadorCliente::vaciarEscenariosJugados()
{
    bloquearEstado();
    escenariosJugados.clear();
    desbloquearEstado();
}
```

```
#include "server_Server.h"
#include "common_SocketCliente.h"
#include <iostream>
#include <string>
using namespace std;

const char* MENSAJE_CERRAR = "salir";
const char* MENSAJE_CONEXION_PUERTO = "Se conecta al puerto ";
int PUERTO_DEFAULT = 3000;

bool modoDebug;

int main(int argc, char* argv[])
{
    Server server;
    string buffer;
    bool cerrar = false;
    modoDebug = false;
    int puerto;

    //Busca el puerto indicado por línea de comando o utiliza el puerto por default
    if (argc < 2)
        puerto = 3000;
    else
        puerto = atoi(argv[1]);

    if (argc > 2)
        if (strcmp("paquetes", argv[2]) == 0)
            modoDebug = true;

    if (server.inicializar(puerto))
        cout << MENSAJE_CONEXION_PUERTO << puerto << endl;
    else
        cerrar = true;

    //Inicia el Servidor
    server.start();

    //Espera el mensaje de cerrar el servidor
    while (!cerrar)
    {
        getline(cin,buffer,'\n');

        if (buffer.compare(MENSAJE_CERRAR) == 0)
            cerrar = true;
        sleep(0);
    }

    //Hace terminar al Servidor enviando una conexión nueva que destrabe el accept
    server.terminar();
    if (server.estaCorriendo())
    {
        SocketCliente mensajeFin;
        mensajeFin.create();
        mensajeFin.connect("127.0.0.1",puerto);
    }
    server.join();
}
```

```
#include "server_Mensaje.h"
#define MENSAJE_INVALIDO 0

Mensaje::Mensaje(std::string mensaje, std::string nombre)
{
    this->nombre = nombre;
    if (mensaje.length() == 0)
    {
        tipo = MENSAJE_INVALIDO; //Invalido
        texto = "";
    }
    else
    {
        tipo = mensaje[0];
        texto = mensaje.substr(1,mensaje.length()-1);
    }
}

bool Mensaje::esValido()
{
    return (tipo != MENSAJE_INVALIDO);
}

std::string& Mensaje::getNombre()
{
    return nombre;
}

char Mensaje::getTipo()
{
    return tipo;
}

std::string& Mensaje::getTexto()
{
    return texto;
}
```

```
#include "server_Server.h"
#include "server_AdministradorUsuarios.h"

#define RUTA_PUNTAJES "puntajes.txt"
#define RUTA_ESCENARIOS "escenarios.txt"

void Server::run()
{
    Socket usuarioNuevo;
    string nombre;
    AdministradorUsuarios admin(RUTA_PUNTAJES,RUTA_ESCENARIOS);

    //Inicia al Administrador de Usuarios
    admin.start();

    while (estaVivo())
    {
        //Espera una nueva conexión
        if (!receptor.accept(usuarioNuevo)) cout << "Falla accept" << endl;

        if (!estaVivo())
            break;

        //Espera nombre
        if (usuarioNuevo.recv(nombre))
        {

            if (nombre[0] == 'E')
            {
                //Usuario existente

                //Analiza si efectivamente existe
                if (!admin.usuarioExiste(nombre.substr(1,nombre.length()-1)))
                {
                    usuarioNuevo.send("SNo existe el usuario");
                    usuarioNuevo.close();
                }

                else
                {
                    if (admin.conectarJugador(nombre.substr(1,nombre.length()
() -1),usuarioNuevo))
                        usuarioNuevo.send("SSe ha conectado exitosamente");

                    else
                    {
                        usuarioNuevo.send("SYa está jugando ese usuario");
                        usuarioNuevo.close();
                    }
                }
            }

            if (nombre[0] == 'N')
                //Usuario nuevo

                //Analiza si lo puede agregar
                if (!admin.agregarUsuario(nombre.substr(1,nombre.length()-1)))
                {
                    usuarioNuevo.send("SUusuario existente");
                    usuarioNuevo.close();
                }

                else
                {
                    admin.conectarJugador(nombre.substr(1,nombre.length()
() -1),usuarioNuevo);
                    usuarioNuevo.send("SSe ha conectado exitosamente");
                }
            }
        }
    }
}
```

```
//Termina al administrador
admin.terminar();
admin.join();
this->setCorriendo(false);

}

bool Server::inicializar(const int puerto)
{
    if (!receptor.create())
    {
        cout << "Falla create" << endl;
        return false;
    }

    if (!receptor.bind(puerto))
    {
        cout << "Falla bind" << endl;
        return false;
    }

    if (!receptor.listen())
    {
        cout << "Falla listen" << endl;
        return false;
    }
    return true;
}
```

```
#include "server_SocketServidor.h"
#include <iostream>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <string>
using namespace std;

bool SocketServidor::bind(const int puerto)
{
    int id = getId();
    sockaddr_in& dir = getDir();

    if (id == SOCKET_INVALIDO)
        return false;

    //Prepara la dirección
    dir.sin_family = AF_INET;
    dir.sin_addr.s_addr = INADDR_ANY;
    dir.sin_port = htons (puerto);

    //Realiza el bind y analiza el resultado
    if (::bind(id,(struct sockaddr*) &dir, sizeof(dir)) == -1)
        return false;
    return true;
}

bool SocketServidor::listen() const
{
    int id = getId();

    if (id == SOCKET_INVALIDO)
        return false;

    if (::listen(id,BACKLOG) == -1)
        return false;
    return true;
}

bool SocketServidor::accept(Socket& nuevo)
{
    int id = getId();
    sockaddr_in& dir = getDir();

    if (id == SOCKET_INVALIDO)
        return false;

    int idNuevo,longDir;
    longDir = sizeof (dir);

    //Obtiene un nuevo id.
    idNuevo = ::accept(id,(struct sockaddr*) &dir, (socklen_t*)&longDir);

    if (idNuevo == -1)
        return false;
    nuevo.setId(idNuevo);
    return true;
}
```

```
#include "StringEnter0.h"

void StringEnter0::enteroAString(std::string& cadena, ulong entero)
{
    if (entero == 0)
        cadena = "0";
    else
        cadena = "";
    while (entero != 0)
    {
        switch (entero %10)
        {
            case 0: cadena="0"+cadena; break;
            case 1: cadena="1"+cadena; break;
            case 2: cadena="2"+cadena; break;
            case 3: cadena="3"+cadena; break;
            case 4: cadena="4"+cadena; break;
            case 5: cadena="5"+cadena; break;
            case 6: cadena="6"+cadena; break;
            case 7: cadena="7"+cadena; break;
            case 8: cadena="8"+cadena; break;
            case 9: cadena="9"+cadena; break;
        }
        entero /= 10;
    }
}
```

Código Fuente

Editor

```
#ifndef ANGULO_H_
#define ANGULO_H_
#include "constantes.h"

class Angulo
{
public:
    static void corregir(long* valor);
    /* Pasajes */
    static double aRadianes(long grados);
    static long aGrados(double radianes);
};

#endif /*ANGULO_H_*/
```

```
#ifndef BARRA_H_
#define BARRA_H_

#include "ElementoLongitudinal.h"

class Barra : public ElementoLongitudinal {

public:
    Barra();
    Barra(Coordenada posicion, ulong magnitud, long angulo, long ancho);
    virtual ~Barra();
};

#endif /*BARRA_H_*/
```

```
#ifndef BOTON_H_
#define BOTON_H_

#include "PanelContenedor.h"
#include "Mapa.h"
#include <string>

class Boton : public PanelContenedor
{
private:
    /* Rutinas */
    void (*pf_rutinaMouseDown)(Coordenada, Boton * );
    void (*pf_rutinaMouseUp)(Coordenada, Boton * );
    void (*pf_rutinaMouseOver)(Coordenada, Boton * );

    /* Buscadores de imagenes */
    std::string getRutaImgDefecto(int nroDeBoton);
    std::string getRutaImgMouseDown(int nroDeBoton);
    std::string getRutaImgMouseOver(int nroDeBoton);

    /* Referencia al mapa sobre el cual interactua */
    Mapa * mapa;
    /* Indica si esta presionado */
    bool presionado;

public:
    /* Constructores y destructores */
    Boton(int nroDeBoton, Mapa *);
    virtual ~Boton();
    /* Eventos del mouse */
    void evento_MouseDown(Coordenada punto);
    void evento_MouseUp(Coordenada punto);
    void evento_MouseOver(Coordenada punto);
    /* Setters de eventos */
    void setRutinaMouseDown(void (*rutina)(Coordenada, Boton *));
    void setRutinaMouseUp(void (*rutina)(Coordenada, Boton *));
    void setRutinaMouseOver(void (*rutina)(Coordenada, Boton *));
    /* Setters, getters y Envios de seniales */
    bool esta_presionado() const;
    void desapretar();
    void enviarSenialSalida();
    void enviarSenialSms();
    void enviarSenialSimulacion();
    long pedirTiempoResolucion();
    bool getSmsState();
    Mapa * getMapa() const;
};

#endif /*BOTON_H_*/
```

```
#ifndef CINTA_H_
#define CINTA_H_

#include "ElementoLongitudinal.h"

class Cinta : public ElementoLongitudinal {
public:
    Cinta();
    Cinta(Coordenada posicion, ulong magnitud, long angulo, long ancho);
    virtual ~Cinta();
};

#endif /*CINTA_H_*/
```

```
#ifndef COHETE_H_
#define COHETE_H_

#include "ElementoLongitudinal.h"

class Cohete : public ElementoLongitudinal
{
public:
    Cohete();
    Cohete(Cordenada posicion, ulong magnitud, long angulo, long ancho);
    virtual ~Cohete();
};

#endif /*COHETE_H_*/
```

```
#ifndef __MUTEX_H__
#define __MUTEX_H__

#include <pthread.h>

class Mutex
{
    private:
        pthread_mutex_t mut;

        /* Constructor copia privado */
        Mutex(const Mutex& m);

        /* Operador = privado */
        Mutex& operator = (const Mutex& original);

    public:
        /* Constructor */
        Mutex();

        /* Lockea el mutex */
        void lock();

        /* Deslockea el mutex */
        void unlock();
};

#endif
```

```
#ifndef __THREAD_H__
#define __THREAD_H__

#include <pthread.h>
#include "common_Mutex.h"

class Thread
{
    private:
        pthread_t hilo;
        Mutex mHilo; //Evita que se lean y escriban los atributos al mismo tiempo.
        bool vivo; //True si tiene que seguir corriendo
        bool corriendo; //True si el hilo está corriendo

        /* Constructor copia privado */
        Thread(const Thread& t);

        /* Operador = privado */
        Thread& operator = (const Thread& original);

    public:
        /* Constructor */
        Thread();

        /* Destructor */
        virtual ~Thread();

        /* Le indica al hilo que debe suicidarse */
        void terminar();

        /* Devuelve true si el hilo aún está corriendo */
        bool estaCorriendo();

        /* Método estático que se empieza a correr al crearse el hilo de tipo pthread */
        static void* static_run (void* p);

        virtual void run() = 0;

        /* Espera que el hilo termine su ejecución */
        void join() const;

        /* Devuelve true si el hilo debe seguir vivo */
        bool estaVivo();

        /* Método que inicia la ejecución del hilo */
        void start();

    protected:
        /* Bloquear y desbloquear estado se utilizan para evitar que se acceda
         * a un atributo del hilo al mismo tiempo (desde el hilo mismo y desde el
         * hilo que lo creó */
        void bloquearEstado();
        void desbloquearEstado();

        /* Modifica el valor de corriendo */
        void setCorriendo(bool valor);

};

#endif
```

```
#ifndef CONFIG_FILE_H
#define CONFIG_FILE_H

#include "config.h"
#include <map>

class ConfigFile : public Config
{
    typedef std::map<std::string, std::string> TStrStrMap;
    TStrStrMap data_;

protected:
    virtual std::string readString(const std::string& key) const;

public:
    ConfigFile(const std::string& filename);
    virtual ~ConfigFile();
};

#endif
```

```
#ifndef CONFIG_H
#define CONFIG_H

#include <sstream>

class Config
{
protected:
    Config() {}
    virtual std::string readString(const std::string& key) const = 0;

public:
    virtual ~Config() {}

    template <typename T>
    T read(const std::string& key) const
    {
        std::istringstream iss(readString(key));
        T ret;
        iss >> ret;

        return ret;
    }
};

#endif
```

```
/* Seteos Ventana */
#define ANCHO_VENTANA 1024
#define ALTO_VENTANA 600

/* Elementos */
#define CANTIDAD_ELEMENTOS 8

#define ELEMENTO_A_BORRAR -2
#define NINGUN_ELEMENTO -1
#define MASA 0
#define PUNTO_FIJO 1
#define RUEDA 2
#define BARRA 3
#define PLATAFORMA 4
#define SOGA 5
#define CINTA 6
#define COHETE 7
#define BORRAR_TODO 8
#define BORRAR_ELEMENTO 9
#define CIRCULAR 10
#define LONGITUDINAL 11
#define ENLAZABLE 12
#define PUNTO_ENLACE 13
#define ZONA_LLEGADA 16
#define GUARDAR 15
#define VOID 17

/* Texturas para paneles */
#define TEXTURA_DEFECTO 0
#define TEXTURA_MOUSEDOWN 1
#define TEXTURA_MOUSEOVER 2
#define TEXTURA_INHABILITADO 3
/* Texturas para el mapa */
#define TEXTURA_MASA 4
#define TEXTURA_PUNTO_FIJO 5
#define TEXTURA_RUEDA 6
#define TEXTURA_BARRA 7
#define TEXTURA_PLATAFORMA 8
#define TEXTURA_SOYA 9
#define TEXTURA_CINTA 10
#define TEXTURA_COHETE 11
#define PLAY 12
#define SALIR 13
#define SMS 14
#define TEXTURA_ZONA_LLEGADA 15

/* Seteos graficos */
#define POLY_SIDES 12

/* Tipos de seleccion de elemento */
#define ELEMENTO_NO_SELECCIONADO 0
#define ELEMENTO_SELECCIONADO POR_EXTREM01 1
#define ELEMENTO_SELECCIONADO POR_EXTREM02 2
#define ELEMENTO_SELECCIONADO POR_CENTRO 3
#define MARGEN_DE_AGARRE 20
#define MARGEN_DE_AGARRE_ANCHO 5
#define LONGITUD_MINIMA 40
#define LONGITUD_MAXIMA 100
#define LONGITUD_COHETE 100
#define PI 3.1415926535897932384626433832795
```

```
#ifndef CONSTNICO
#define CONSTNICO

/* Seteos Tiempo */
#define TIEMPOGANAR 4000.0
#define TIME_C 1000.0

#define CLOCK_X -1.6
#define CLOCK_Y -0.9

/* Definiciones Físicas */
#define RADIOMASA 2.0/30.0
    const double LONA_K = 40000;
    const double METAL_K = 50000;
    const double METAL_WIDTH=0.07;
    const double RUEDA_K = 10000;
    const long ROCKET_K=80000;
    const double ROCKET_WIDTH=0.07;
    const double ROPE_K = 10000;
    const double BALL_K = 10000;

#endif
```

```
#ifndef COORDENADA_H_
#define COORDENADA_H_
#include <string>
#include "Angulo.h"
typedef unsigned long ulong;

class Coordenada {
public:
    /* Atributo publico X */
    long x;
    /* Atributo publico Y */
    long y;
    /* Atributo publico modulo */
    ulong modulo;
    /* Atributo publico angulo */
    long angulo;
    /* Contructor vacio */
    Coordenada();
    /* Contructor con parametros x e y */
    Coordenada(long x, long y);
    /* Contructor con parametros modulo y angulo */
    Coordenada(ulong modulo, long angulo);
    /* Destructor virtual */
    virtual ~Coordenada();
    /* Se modifica en delta */
    void modificarDelta(Cordenada);
    /* Mueve a otra coordenada */
    void modificar(Cordenada);
    /* Setea X, modifica angulo y modulo */
    void setX(long x);
    /* Setea Y, modifica angulo y modulo */
    void setY(long y);
    /* Setea angulo, modifica X e Y */
    void setAngulo(long angulo);
    /* Setea modulo, modifica X e Y */
    void setModulo(ulong modulo);
    /* Imprime por pantalla el valor */
    void imprimir() const;
    /* Operador Suma */
    Coordenada operator+(const Coordenada) const;
    /* Operador Resta */
    Coordenada operator-(const Coordenada) const;

private:
    void actualizarModulo();
    void actualizarAngulo();
    void actualizarX();
    void actualizarY();

};

#endif /*COORDENADA_H_*/
```

```
#include <time.h>
#include <ctime>
class Cronometro{
public:
    void iniciar();
    void detener();
    long getMilisegundos();
    int estaParado();

private:
    int parado;
    clock_t final, comienzo;
};
```

```
#ifndef ELEMENTO_H
#define ELEMENTO_H
#include <vector>

class Element
{
public:
    Element(unsigned f, unsigned l);
    unsigned getFirst();
    unsigned getLast();
    std::vector<int> masas;
private:
    unsigned first;
    unsigned last;
};

#endif
```

```
#ifndef ELEMENTMANAGER_H
#define ELEMENTMANAGER_H

#include <cmath>
#include <iostream>
#include <vector>
#include <map>
#include <utility>
#include "element.h"
#include "zonaLlegada.h"

class ElementManager
{
public:
    ElementManager();
    ~ElementManager();
    void addRope(int first, int last);
    void addLona(int first, int last);
    void addMetalBar(int first, int last);
    void addPlataforma(int first, int last);
    void addCohete(int first, int last);
    void addRueda(int first, int last);
    void addMainBall(int first,int last);
    void addPuntoFijo(int first,int last);
    void addZonaLlegada(double x,double y,double width,double heighth);
    void addSpringElement(int spring, int element);
    std::vector<Element*> getRopes();
    std::vector<Element*> getRuedas();
    std::vector<Element*> getMetalBars();
    std::vector<Element*> getPlataformas();
    std::vector<Element*> getLonas();
    std::vector<Element*> getCohetes();
    std::vector<Element*> getMainBalls();
    std::vector<Element*> getPuntosFijos();
    std::vector<ZonaLlegada*> getZonasLlegada();
    std::vector< std::vector<Element*>> vectoresElementos;
    std::map<int,int> getSpringElements();

private:
    std::vector<Element*> ropes;
    std::vector<Element*> cohetes;
    std::vector<Element*> ruedas;
    std::vector<Element*> plataformas;
    std::vector<Element*> metalBars;
    std::vector<Element*> mainBalls;
    std::vector<Element*> lonas;
    std::vector<Element*> puntosFijos;
    std::vector<ZonaLlegada*> zonasLlegada;
    std::map<int,int> springElements;

};

#endif
```

```
#ifndef ELEMENTO_H_
#define ELEMENTO_H_

#include <iostream>
#include "Coordenada.h"
#include <sstream>
#include "constantes.h"

typedef unsigned long ulong;

class Elemento {
private:
    ulong magnitud; /* Magnitud del elemento. Se adecua al tipo de elemento */
    long angulo; /* Angulo del elemento */
    Coordenada posicion; /* Posicion del punto representativo del elemento */
    bool bloqueado; /* Indica si el elemento puede ser modificado */
    std::string nombre; /* Nombre y jerarquia del elemento */
    bool mostrarEnlaces; /* Si debe mostrar sus enlaces */

public:
    /* Contructor vacio */
    Elemento();
    /* Contructor con parametros */
    Elemento(const std::string & nombre, Coordenada posicion,
             const ulong magnitud, long angulo);
    /* Contructor virtual */
    virtual ~Elemento();

    /* Redimensiona la longitud en deltaUnidades. */
    void setMagnitud(const long valor);
    /* Devuelve el valor de la magnitud del elemento */
    ulong getMagnitud() const;
    /* Da un valor al angulo */
    virtual void setAngulo(const long);
    /* Devuelve el angulo */
    long getAngulo() const;
    /* Guarda la posicion en la que se encuentra el Elemento */
    virtual void setPosicion(const Coordenada nuevaPosicion);
    /* Devuelve la posicion en la que se encuentra el Elemento */
    Coordenada getPosition() const;
    /* Devuelve el extremo opuesto calculado por trigonometria */
    virtual Coordenada getExtremoOpuesto() const;
    /* Bloquea al elemento: no se puede mover, redimensionar ni borrar */
    void bloquear();
    /* True si no es movable, rotable ni redimensionable */
    bool estaBloqueado() const;
    /* Devuelve el nombre del elemento en un string */
    std::string getNombre() const;
    /* Dice si es un determinado elemento, pregunta a getNombre */
    bool es(int const nro_elemento) const;
    /* Devuelve el tipo de elemento */
    int getTipo() const;
    /* Devuelve true si la Coordenada pertenece al Elemento */
    virtual int seleccionar(Coordenada) const = 0;
    /* Imprime por stdout la informacion del elemento */
    virtual void imprimir();
    /* Dice si deben aparecer los puntos de enlace */
    bool getMostrarPuntosDeEnlace() const;
    /* Asigna si deben aparecer los puntos de enlace */
    void setMostrarPuntosDeEnlace(const bool);

};

#endif /*ELEMENTO_H_*/
```

```
#ifndef ELEMENTOCIRCULAR_H_
#define ELEMENTOCIRCULAR_H_

#include "Elemento.h"
#include <string>

class ElementoCircular : public Elemento {

public:
    ElementoCircular();
    ElementoCircular(const std::string & nombre, Coordenada posicion,
                    const ulong magnitud, long angulo);
    virtual ~ElementoCircular();

    /* Devuelve true si la Coordenada pertenece al Elemento */
    virtual int seleccionar(Coordenada) const;
};

#endif /*ELEMENTOCIRCULAR_H_*/
```

```
#ifndef ELEMENTOENLAZABLE_H_
#define ELEMENTOENLAZABLE_H_

#include "ElementoCircular.h"
#include "Elemento.h"
#include "Coordenada.h"
#include <list>

class ElementoEnlazable : public ElementoCircular
{
private:
    Elemento * poseedor;
    std::list < ElementoEnlazable* > _enlaces;

public:
    ElementoEnlazable(const std::string & nombre,
                      Coordenada posicion, const ulong magnitud, long angulo, Elemento * poseedor);
    virtual ~ElementoEnlazable();

    /* Devuelve el Elemento que lo contiene */
    Elemento * getPoseedor() const;
    /* Devuelve la lista de enlazables que tiene asociados */
    std::list <ElementoEnlazable*> getListaEnlaces() const;
    /* Vacia los enlaces y los libera */
    void eliminarEnlaces();
    /* Agrega un enlace a la lista verificando y devuelve si pudo
     hacerlo (verifica que su posicion sea igual que la propia)
     */
    bool conectarA(ElementoEnlazable* );
    bool enlazado;

};

#endif /*ELEMENTOENLAZABLE_H_*/
```

```
#ifndef ELEMENTOLONGITUDINAL_H_
#define ELEMENTOLONGITUDINAL_H_

#include "Elemento.h"
#include <string>
#include <list>
#include "PuntoDeEnlace.h"

#define LONGITUD_DE_ENLACE_MINIMA 50

class ElementoLongitudinal : public Elemento {

private:
    std::list <PuntoDeEnlace*> _enlaces;
    Coordenada extremoOpuesto;

protected:
    ulong ancho;

public:
    ElementoLongitudinal();
    ElementoLongitudinal(const std::string & nombre, Coordenada posicion,
                         const ulong magnitud, long angulo, ulong ancho);
    virtual ~ElementoLongitudinal();

    /* Agrega a la lista los puntos de enlaces en funcion de la longitud
     * determinando su posicion */
    void actualizarPuntosDeEnlace(int modo);
    /* Imprime los enlaces */
    void imprimirEnlaces();
    /* Devuelve puntero a lista de enlaces */
    std::list<PuntoDeEnlace*> getListadeEnlaces();
    /* Elimina los enlaces */
    void eliminarEnlaces();
    /* Devuelve ancho */
    ulong getAncho() const;
    /* Setea el extremo opuesto y actualiza el modulo y angulo */
    void setExtremoOpuesto(const Coordenada);
    /* Devuelve el extremo Opuesto */
    Coordenada getExtremoOpuesto() const;
    /* Devuelve true si la Coordenada pertenece al Elemento */
    virtual int seleccionar(Coordenada) const;
    /* Redefino setPosicion. Actualizo extremoOpuesto */
    void setPosicion(const Coordenada nuevaPosicion);
    /* Redefino setAngulo. Actualizo extremoOpuesto */
    void setAngulo(const long angulo);

};

#endif /*ELEMENTOLONGITUDINAL_H_*/
```

```
#ifndef FABRICADEELEMENTOS_H_
#define FABRICADEELEMENTOS_H_

#include "Coordenada.h"
#include "Elemento.h"
#include "Masa.h"
#include "Rueda.h"
#include "PuntoFijo.h"
#include "Barra.h"
#include "Soga.h"
#include "Plataforma.h"
#include "Cinta.h"
#include "Cohete.h"
#include "constantes.h"
#include "Llegada.h"

class FabricaDeElementos
{
public:
    FabricaDeElementos();
    virtual ~FabricaDeElementos();
    Elemento * crear (int nroElemento, Coordenada posicion);
};

#endif /*FABRICADEELEMENTOS_H_*/
```

```
// Coded by Mariano M. Chouza (http://www.chouza.com.ar) replacing a version
// present in "Sol's 2d gl basecode 2.0" (http://www.iki.fi/sol/)

#ifndef FILEUTILS_H
#define FILEUTILS_H

#include <fstream>
#include <SDL.h>

class BinFile
{
    std::fstream s_;
public:
    BinFile(const char *filename);
    virtual ~BinFile();
    char readbyte();
    int16_t readword();
    int32_t readdword();
    void readbytes(char* buffer, size_t num);
    size_t tell();
    void seek(size_t pos);
};

#endif
```

```
#ifndef __HILO_CRON_H__
#define __HILO_CRON_H__

#include "common_Thread.h"
#include <iostream>
#include "cronometro.h"

class HiloCron: public Thread
{
private:
    Cronometro* cronometro;

public:
    /* Constructor */
    HiloCron();

    /* Destructor */
    virtual ~HiloCron();

    long getMilisegundos();

    /* Función que ejecuta el hilo. */
    void run();
};

#endif
```

```
#ifndef INTERFAZEDICION_H_
#define INTERFAZEDICION_H_


#include <iostream>
#include "Angulo.h"
#include <string>
#include "Mapa.h"
#include "Masa.h"
#include "Barra.h"
#include "Ventana.h"
#include "PanelContenedor.h"
#include "Boton.h"
#include "InterfazMapa.h"
#include "constantes.h"
#include "Elemento.h"

/* Eventos sobre el mapa */
void mapa_mouseDown(Coordenada punto, InterfazMapa *);
void mapa_mouseOver(Coordenada punto, InterfazMapa *);
void mapa_mouseUp(Coordenada punto, InterfazMapa *);
/* Eventos de la botonera */
void botonMasa_mouseDown(Coordenada cord, Boton *);
void botonRueda_mouseDown(Coordenada cord, Boton *);
void botonPuntoFijo_mouseDown(Coordenada cord, Boton * boton);
void botonBarra_mouseDown(Coordenada cord, Boton *);
void botonPlataforma_mouseDown(Coordenada cord, Boton * boton);
void botonSoga_mouseDown(Coordenada cord, Boton * boton);
void botonCinta_mouseDown(Coordenada cord, Boton * boton);
void botonZonaLlegada_mouseDown(Coordenada cord, Boton * boton);
void botonCohete_mouseDown(Coordenada cord, Boton * boton);
void botonBorrarTodo_mouseDown(Coordenada cord, Boton * boton);
void botonBorrarElemento_mouseDown(Coordenada cord, Boton * boton);
void botonPlay_mouseDown(Coordenada cord, Boton * boton);
void botonSalir_mouseDown(Coordenada cord, Boton * boton);
void botonSms_mouseDown(Coordenada cord, Boton * boton);
void botonGuardar_mouseDown(Coordenada cord, Boton * boton);

class InterfazEdicion {

private:
    /* Referencia al modelo del mapa sobre el cual editara */
    Mapa * mapa;
    /* Agrega los botones */
    void agregarBotoneraInferior(Ventana *);

public:
    InterfazEdicion(Mapa*);
    ~InterfazEdicion();
    void comenzar();

};

#endif /*INTERFAZEDICION_H_*/
```

```
#ifndef INTERFAZMAPA_H_
#define INTERFAZMAPA_H_

#include "Coordenada.h"
#include "PanelContenedor.h"
#include "Mapa.h"
#include <string>
#include "ElementoCircular.h"
#include "ElementoLongitudinal.h"
#include "Llegada.h"
class InterfazMapa : public PanelContenedor
{
private:
    /* Visibilidad del modelo mapa */
    Mapa * mapa;
    /* Rutinas del mouse */
    void (*pf_rutinaMouseDown)(Coordenada, InterfazMapa *);
    void (*pf_rutinaMouseUp)(Coordenada, InterfazMapa *);
    void (*pf_rutinaMouseOver)(Coordenada, InterfazMapa *);

    /* Metodos para imprimir cada tipo de elemento */
    void imprimirElementoCircular(ElementoCircular * circular);
    void imprimirElementoLongitudinal(ElementoLongitudinal * longitudinal);
    void imprimirZonaLlegada(Llegada * llegada);

public:
    /* Constructores y destructores, con y sin zoom de vision */
    InterfazMapa(const std::string & rutaImgDefecto, const std::string & rutaImgClickDown, Mapa * mapa);
    InterfazMapa(const std::string & rutaImgDefecto, const std::string & rutaImgClickDown, Mapa * mapa, double zoom);
    virtual ~InterfazMapa();

    /* Redefinición de llamadas a eventos */
    void evento_MouseDown(Coordenada punto);
    void evento_MouseUp(Coordenada punto);
    void evento_MouseOver(Coordenada punto);

    /* setters para las rutinas de cada evento */
    void setRutinaMouseDown(void (*rutina)(Coordenada, InterfazMapa *));
    void setRutinaMouseUp(void (*rutina)(Coordenada, InterfazMapa *));
    void setRutinaMouseOver(void (*rutina)(Coordenada, InterfazMapa *));
    Coordenada vectorDiferencia;
    /* getter y setter del mapa */
    Mapa * getMapa() const;
    void setMapa(Mapa* mapa); //destruye el viejo
    /* Dibuja el mapa */
    void imprimir();

};

#endif
```

```
#ifndef LLEGADA_H_
#define LLEGADA_H_

#include <iostream>
#include "Coordenada.h"
#include "constantes.h"
#include "Elemento.h"

class Llegada : public Elemento {

public:
    Llegada(Coordenada);
    int seleccionar(Coordenada) const;

};

#endif /*LLEGADA_H_*/
```

```
#ifndef TEST_SMS_H
#define TEST_SMS_H

#include "spring_mass_system.h"

void load(SpringMassSystem& sms);

#endif
```

```
#ifndef MAPA_H_
#define MAPA_H_
#include "Elemento.h"
#include <list>
#include "FabricaDeElementos.h"
#include "constantes.h"
typedef unsigned long ulong;

class Mapa {
private:
    /* Nombre del mapa. Sirve para buscar el archivo de imagen de fondo */
    std::string nombre;
    /* Ancho y alto del mapa */
    ulong ancho;
    ulong alto;
    /* Lista de elementos contenidos en mapa */
    std::list<Elemento*> _elementos;
    /* Es el elemento seleccionado. Todos los metodos trabajaran sobre el */
    Elemento * elemento;
    /* Es el elemento que se creará al llamar agregarElemento() */
    int nroElementoParaCrear;
    /* Indica por donde se a seleccionado el elemento */
    int tipoSeleccion;
    /* Vector de elementos habilitados */
    bool elementosHabilitados [CANTIDAD_ELEMENTOS];
    /* Indica los precios de los elementos de este mapa en particular */
    unsigned int precio[CANTIDAD_ELEMENTOS];
    /* Indica la plata que este mapa provee */
    int plata;
    /* Indica el puntaje */
    int puntos;
    int tiempoCohete; // milisegundos. 0 es tiempo infinito
    long tiempoResolucion; // milisegundos de tiempo de resolucion del mapa
    /* Para determinar si el mapa se debe guardar o no */
    bool aGuardar;
public:
    /* Contructor con valores por default */
    Mapa();
    /* Contructor con parametros */
    Mapa(ulong ancho, ulong alto);
    /* Destructor virtual */
    virtual ~Mapa();
    /* Devuelve ancho */
    ulong getAncho() const;
    /* Devuelve alto */
    ulong getAlto() const;
    /* Setea el elemento que la fabrica contruira cuando se lo pida */
    void setNroElementoParaCrear(int);
    /* Devuelve el elemento a crear */
    int getNroElementoParaCrear() const;
    /* Agrega elemento al mapa. Devuelve true si fue posible */
    bool agregarElemento(Coordenada posicion);
    /* Selecciona el primer elemento que encuentre en esa coordenada.
     * Devuelve False en caso de no haber ninguno
     * */
    int seleccionarElemento(Coordenada);
    /* Informa si hay algun elemento seleccionado */
    bool hayElementoSeleccionado() const;
    /* Devuelve por donde se ha seleccionado el elemento */
    int getTipoSeleccion() const;
    /* setea el tipo de seleccion de forma forzada */
    void setTipoSeleccion(const int tipo);
    /* Devuelve puntero constante al Elemento seleccionado */
    const Elemento * getElementoSeleccionado() const;
    /* Devuelve cantidad de elementos en el mapa */
    size_t getCantidadElementos();
    /* Devuelve true si existe punto esta en el entorno
     * de algun punto de enlace.
     * Retorna en puntoExacto el centro del punto de enlace
     * */
    bool entornoDePuntoDeEnlace(Coordenada punto, Coordenada * puntoExacto);
    /* Setea el extremo opuesto del elemento longitudinal seleccionado */
}
```

```
void setExtremoOpuesto(const Coordenada punto);
/* Devuelve el extremo opuesto del elemento longitudinal seleccionado */
Coordenada getExtremoOpuesto() const;
/* Mueve el elemento a destino. Devuelve true si fue posible */
bool moverElementoA(Coordenada posicion);
/* Rota el elemento a destino. Devuelve true si fue posible */
bool rotarElementoA(long angulo);
/* Agranda el elemento a longitud. Devuelve true si fue posible */
bool agrandarElementoA(long longitud);
/* Actualiza modulo y angulo en funcion del extremoOpuesto.
 * Mantiene posicion constante.
 * Retora la coordenada corregida.
 */
Coordenada actualizarModuloAngulo(Coordenada extremoOpuesto);
/* Actualiza solo el angulo */
Coordenada actualizarAngulo(Coordenada extremoOpuesto);
Coordenada actualizarAngulo2(Coordenada click);
/* Elimina el elemento y extrae de la lista*/
void eliminarElemento();
/* Elimina todos y vacia la lista */
void eliminarTodosLosElementos();
/* Guarda en los buffers posicion, angulo y magnitud del Elemento */
void guardarConfiguracionElemento();
/* Mueve el elemento a su posicion, angulo y magnitud de origen */
void re establecerConfiguracionElemento();
/* Imprime todos los elementos */
void imprimir();
/* Devuelve la lista de elementos */
std::list <Elemento*> getListadeElementos() const;
/* Actualiza los puntos de enlace de los elementos longitudinales */
void actualizarEnlaces();
/* Si el elemento seleccionado debera ser elmininado */
bool elementoABorrar;
/* Filtra los enlaces en todo el mapa y los del elementos longitudinales */
std::list <ElementoEnlazable*> getTodosLosEnlaces();
/* Avisa al Elemento Seleccionado si mostrar sus puntos de enlace */
void setMostrarPuntosDeEnlaceElemento(const bool);
/* Bloquea todos los elementos del mapa */
void bloquearElementos();
/* Deshabilita un elemento */
void deshabilitarElemento(const int nroElemento);
/* Habilita un elemento */
void habilitarElemento(const int nroElemento);
/* Devueleve si el elemento esta habilitado */
bool elementoHabilitado(const int nroElemento) const;
/* Muestra por consola los enlaces */
void imprimirTodosLosEnlaces();
/* Chequea si el escenario es valido: necesita una Masa y una Llegada */
bool esCorrecto();

/* Otros getters y setter de menor importancia */
void setNombre(const std::string&);
std::string getNombre() const;
void setPlata(const int);
int getPlata() const;
void setPuntos(const int);
int getPuntos() const;
void setGuardar(bool valor);
bool debeGuardarse();
void setPrecio(int nroElemento, int precio);
int getPrecio(int nroElemento) const;
int getTiempoCohete() const; // milisegundos
void setTiempoCohete(int tiempo); // milisegundos
};

#endif /*MAPA_H_*/
```

```
#ifndef MASA_H_
#define MASA_H_

#include "ElementoCircular.h"

class Masa : public ElementoCircular {
public:
    /* Constructor por defecto */
    Masa();
    /* Constructor con parametros */
    Masa(Coordenada posicion, ulong magnitud);
    /* Destructor virtual */
    virtual ~Masa();

};

#endif /*MASA_H_*/
```

```
#ifndef PANELCONTENEDOR_H_
#define PANELCONTENEDOR_H_
#include "SDL.h"
#include "SDL_image.h"
#include <SDL_opengl.h>
#include "Coordenada.h"
#include <list>
#include <map>
#include <string>
#include <math.h>
#include "Ventana.h"
#include "constantes.h"

class Ventana;

typedef unsigned long ulong;

class PanelContenedor
{
private:
    /* ancho pantalla */
    ulong ancho;
    /* alto pantalla */
    ulong alto;
    /* Posicion del panel en pantalla */
    Coordenada posicion;
    /* numero de textura que se imprime actualmente */
    int texturaActual;
    /* Referencia a la pantalla */
    SDL_Surface * pantalla;
    /* Visibilidad de la textura */
    bool visible;
    /* Lista de paneles contenidos */
    std::list<PanelContenedor*> _paneles;
    /* Indica si fue clickeado */
    bool clicked;
    /* Indica si tiene el mouse arriba */
    bool mouseOver;
    /* Nombre en la jerarquia */
    std::string nombre;
    /* Id de panel */
    int id;

    ulong getAncho(const std::string & rutaImg);
    ulong getAlto(const std::string & rutaImg);

protected:
    /* mapa de texturas */
    std::map<int, GLuint> _texturas;
    /* Referencia de la ventana contenedora */
    Ventana * ventana;
    /* Carga una textura al map */
    void loadTexture(const char * path, int tipoTextura);
    /* Indica si esta habilitado */
    bool habilitado;

public:
    /* Constructores y destructor */
    PanelContenedor(const std::string & nombre, const std::string & rutaImgDefecto, const std::string & rutaImgClickDown, const std::string & rutaImgMouseOver);
    PanelContenedor(const std::string & nombre, const std::string & rutaImgDefecto, const std::string & rutaImgClickDown, const std::string & rutaImgMouseOver, double zoom);
    virtual ~PanelContenedor();
    /* Imprime en pantalla el panel y subpaneles */
    virtual void imprimir();
    /* Imprime la textura actual */
    void imprimirImagenDeFondo();
    /* Agregan paneles a la lista */
    void agregarPanel(PanelContenedor *, Coordenada);
    void agregarPanel(PanelContenedor * panel, int lugar);
```

```
/* Utilitarios para la conversion de cordenadas SDL a OpenGL */
double cambioX(long x);
double cambioY(long y);
/* Devuelve true si la coordenada pertenece al panel */
bool pertenece(Cordenada);
/* Ejecuta evento o bien lo ejecuta en un subpanel */
bool seEjecutaEventoEn(const Cordenada punto, int evento);

//SETTERS Y GETTERS
void setPantalla (SDL_Surface * pantalla);
void setVentana(Ventana *);
void setPosicion(const Cordenada posicion); //la setea el agregador de panel
Cordenada getPosicion() const;
void setImagenDeFondo(int);
void setAltura(const ulong);
void setAncho(const ulong);
ulong getAltura() const;
ulong getAncho() const;
bool is_clicked() const;
bool is_over() const;
std::string getNombre() const;
int getId() const;
void setId(const int);
void setMouseOver(const bool b);
bool estaHabilitado() const;
void setHabilitar(const bool);

//EVENTOS
void virtual evento_MouseDown(Cordenada punto);
void virtual evento_MouseOver(Cordenada punto);
void virtual evento_MouseUp(Cordenada punto);

//VARIABLES DE CONVERSION DE SDL A OPENGL
double TX;
double TY;

};

#endif /*PANELCONTENEDOR_H_*/
```

```
#ifndef __PARSER_XML_H__
#define __PARSER_XML_H__

#include "Mapa.h"
#include "Elemento.h"
#include <string>
#include <libxml/xmlreader.h>

class ParserXML
{
public:
    /* Devuelve un mapa en base a un XML levantado de un archivo */
    Mapa* obtenerMapaArchivo (const std::string& rutaEntrada) const;

    /* Devuelve un mapa en base a un XML que está en memoria y se pasa
     * por parámetro */
    Mapa* obtenerMapaMemoria (const std::string& xml) const;

    /* Se genera un archivo XML en base al mapa dado */
    void obtenerXMLArchivo (const std::string& rutaSalida, const Mapa* mapa) const;

    /* Se genera un XML en Memoria en base a un mapa dado */
    void obtenerXMLMemoria (std::string& xml, const Mapa* mapa) const;

private:
    /* Analiza el contenido de un nodo y en base al nombre del tag invoca al método
     * correspondiente */
    void analizarNodo(Mapa* mapa, xmlTextReaderPtr reader) const;

    /* Convierte un entero a un string */
    void enteroAString(std::string& cadena, ulong entero) const;

    /* Analiza un elemento en su totalidad y lo agrega al mapa */
    void procesarElemento(Mapa* mapa, xmlTextReaderPtr reader) const;

    /* Lee los costos de cada elemento */
    void procesarCostos(Mapa* mapa, xmlTextReaderPtr reader) const;

    /* Analiza qué elementos debe habilitar y cuáles no */
    void procesarHabilitados(Mapa* mapa, xmlTextReaderPtr reader) const;

    /* Imprime qué botones están habilitados y cuales no en el mapa */
    void imprimirHabilitados(std::string& xml, const Mapa* mapa) const;

    /* Imprime todos los elementos del mapa */
    void imprimirElementos(std::string& xml, const Mapa* mapa) const;

    /* Obtiene un mapa a partir del lector dado */
    Mapa* obtenerMapa (xmlTextReaderPtr reader) const;

    /* Imprime el tiempo del cohete */
    void imprimirTiempo(std::string& xml, const Mapa* mapa) const;

    /* Imprime los costos de cada elemento */
    void imprimirCostes(std::string& xml, const Mapa* mapa) const;

    /* Determina el número de elemento según el tipo dado */
    int nroElemento(const std::string& tipo) const;

};

#endif
```

```
#ifndef PLATAFORMA_H_
#define PLATAFORMA_H_

#include "ElementoLongitudinal.h"

class Plataforma : public ElementoLongitudinal
{
public:
    Plataforma();
    Plataforma(Coordenada posicion, ulong magnitud, long angulo, long ancho);
    virtual ~Plataforma();
};

#endif /*PLATAFORMA_H_*/
```

```
#ifndef PUNTODEENLACE_H_
#define PUNTODEENLACE_H_

#include "ElementoEnlazable.h"

class PuntoDeEnlace : public ElementoEnlazable
{
public:
    PuntoDeEnlace();
    PuntoDeEnlace(Coordenada posicion, Elemento * poseedor);
    virtual ~PuntoDeEnlace();
};

#endif /*PUNTODEENLACE_H_*/
```

```
#ifndef PUNTOFIJO_H_
#define PUNTOFIJO_H_

#include "ElementoEnlazable.h"

class PuntoFijo : public ElementoEnlazable
{
public:
    PuntoFijo();
    PuntoFijo(Coordenada posicion, ulong magnitud);
    virtual ~PuntoFijo();
};

#endif /*PUNTOFIJO_H_*/
```

```
#ifndef __RELOJ_H__
#define __RELOJ_H__

#include <string>
#include <map>
#include <SDL.h>
#include "SDL_image.h"
#include <SDL_opengl.h>

class Reloj
{
    private:
        std::map<char,GLuint> texturas;
        void imprimirCaracter(char letra, float x, float y);
        void loadTexture(const char * path, char letra);

    public:
        void imprimirReloj(const std::string& reloj, float x, float y);
        void cargarTexturas();

};

#endif
```

```
#ifndef RUEDA_H_
#define RUEDA_H_

#include "ElementoCircular.h"
#include <list>
#include "PuntoDeEnlace.h"

class Rueda : public ElementoCircular {
private:
    std::list<PuntoDeEnlace*> _enlaces;

public:
    /* Constructor por defecto */
    Rueda();
    /* Constructor con parametros */
    Rueda(Coordenada posicion, ulong magnitud);
    /* Destructor virtual */
    virtual ~Rueda();
    /* Ubica los enlaces */
    void actualizarPuntosDeEnlace();
    /* Imprime los enlaces */
    void imprimirEnlaces();
    /* Devuelve puntero a lista de enlaces */
    std::list<PuntoDeEnlace*> getListaEnlaces();
    /* Elimina los enlaces */
    void eliminarEnlaces();

};

#endif /*RUEDA_H_*/
```

```
#ifndef SMS_APP_H
#define SMS_APP_H

#include "config_file.h"
#include "spring_mass_system.h"
#include "video.h"
#include "hiloCron.h"
#include <string>
#include "Reloj.h"
#include <SDL.h>

class SMSApp
{
    /*Muestra el tiempo de simulación en pantalla*/
    Reloj reloj;
    /*Carga el archivo de configuracion*/
    ConfigFile config_;
    /*Contiene información del sistema de masas y resortes*/
    SpringMassSystem sms_;
    /*Carga y maneja texturas y configuraciones gráficas*/
    Video video_;
    uint32_t ticks_;
    int physMult_;
    /*Dibuja la situación actual del SMR*/
    void drawFrame(std::string & time);
    /*Devuelve true si se cumple el objetivo del juego*/
    bool ganaNivel();
    /*Verifica la posición de la masa principal comparandola con
     las areas de llegada*/
    bool estaEnLlegada();
    bool estabEnLlegada;
    bool erased(int spring);
    /*Thread que controla el tiempo de simulación*/
    HiloCron* tiempoLlegada;
public:
    SMSApp(int argc, Mapa* mapa);
    virtual ~SMSApp();
    int run();
    bool sms;
    int controlLonas;
    std::map<int, int> borrados;
};

#endif
```

```
#ifndef SOGA_H_
#define SOGA_H_

#include "ElementoLongitudinal.h"

class Soga : public ElementoLongitudinal
{
public:
    Soga();
    Soga(Coordenada posicion, ulong magnitud, long angulo, long ancho);
    virtual ~Soga();
};

#endif /*SOGA_H_*/
```

```
#ifndef SPRING_MASS_SYSTEM_H
#define SPRING_MASS_SYSTEM_H
#include <algorithm>
#include "config.h"
#include "vector2.h"
#include "elementManager.h"
#include <utility>
#include <vector>
#include <list>
#include "Mapa.h"

class SpringMassSystem;

class MassProxy
{
    friend class SpringMassSystem;

    SpringMassSystem& SMS_;
    size_t index_;

    MassProxy(SpringMassSystem& sms, size_t index);

public:
    double getInvMass() const;
    const Vector2<>& getPos() const;
    const Vector2<>& getVel() const;
    void setVel(const Vector2<>& v);
};

class SpringProxy
{
    friend class SpringMassSystem;

    SpringMassSystem& SMS_;
    size_t index_;

    SpringProxy(SpringMassSystem& sms, size_t index);

public:
    const Vector2<>& getStartPos() const;
    const Vector2<>& getEndPos() const;
    double getRelElongation() const;
    double getK() const;
};

struct SMSStaticState
{
    // Masses data
    std::vector<double> invMassVec;

    // Springs data
    std::vector<std::pair<int, int> > springEndPointsVec;
    std::vector<double> springKsVec;
    std::vector<double> springNormalLengthsVec;
    std::vector<double> springInvStrength;

    // Gravity
    Vector2<> g;

    // Viscous damping
    double damping;

    // Collidable masses info
    std::vector<int> collMassesIndices;
    std::vector<double> collMassesRadii;
};

class SpringMassSystem
{
    // System dynamic state (the positions are stored in the first half and
    // velocities in the other half)
    std::vector<Vector2<> > y_;
```

```
// System static state (the part of the state that doesn't change)
SMSStaticState x_;

// Friends
friend class MassProxy;
friend class SpringProxy;

ElementManager manager;
Mapa* mapa;
std::vector<int> enganches;

public:
    SpringMassSystem(const Config& config);

    int addMass(double invMass, const Vector2<>& pos);
    int addCollMass(double invMass, const Vector2<>& pos, double radius);
    int addSpring(int startMass, int endMass, double k,
                  double relElongation = 0.0, double invStrength = 0.0);
    void fixMass(int pos);
    MassProxy getMassProxy(size_t index);
    SpringProxy getSpringProxy(size_t index);

    size_t getNumMasses() const;
    size_t getNumSprings() const;

    double getPotentialEnergy() const;
    double getKineticEnergy() const;
    double getEnergy() const;

    void eulerStep(double dt);
    void rk4Step(double dt);

    void debugPrint(std::ostream& os) const;

    bool sameElement(unsigned first,unsigned second);
    void unifyMass(int first,int second);
    void checkRoutine();
    ElementManager* getElementManager();
    std::vector<int> getVecEnganches();
    void addEnganche(int enganche);

    void setMapa(Mapa* mapa);
    Mapa* getMapa();
};

#endif
```

```
#ifndef TEXTURE_STORE_H
#define TEXTURE_STORE_H

#include <map>
#include <string>
#include <SDL_opengl.h>

class TextureStore
{
    typedef std::map<std::string, GLuint> TTexStore;
    static TTexStore texStore_;
public:
    static GLuint addTexture(const std::string& filename);
    static GLuint getTexture(const std::string& filename);
};

#endif
```

```
#ifndef VECTOR2_H
#define VECTOR2_H

#include <cmath>
#include <iostream>

template <typename Real = double>
struct Vector2
{
    Real x;
    Real y;

    Vector2() : x(0), y(0) {}
    Vector2(const Vector2& v) : x(v.x), y(v.y) {}
    Vector2(Real x, Real y) : x(x), y(y) {}

    Vector2& operator+=(const Vector2& v)
    {
        x += v.x;
        y += v.y;
        return *this;
    }

    Vector2& operator-=(const Vector2& v)
    {
        x -= v.x;
        y -= v.y;
        return *this;
    }

    Vector2& operator*=(Real f)
    {
        x *= f;
        y *= f;
        return *this;
    }

    Vector2& normalize()
    {
        return *this *= 1.0 / norm();
    }

    double dot(const Vector2& v) const
    {
        return x * v.x + y * v.y;
    }

    double sqNorm() const
    {
        return x * x + y * y;
    }

    double norm() const
    {
        return sqrt(x * x + y * y);
    }

    friend std::ostream& operator<<(std::ostream& os, const Vector2& v)
    {
        os << v.x << " " << v.y;
        return os;
    }

    friend std::istream& operator>>(std::istream& is, Vector2& v)
    {
        is >> v.x >> v.y;
        return is;
    }
};

#endif
```

```
#ifndef VENTANA_H_
#define VENTANA_H_

#include <SDL.h>
#include <SDL_image.h>
#include "PanelContenedor.h"
#include "Coordenada.h"
#include <list>
#include <SDL_opengl.h>
#include "constantes.h"

class PanelContenedor;

class Ventana
{
private:
    /* Pantalla grafica */
    SDL_Surface * pantalla;
    /* Dimensiones pantalla */
    int ancho;
    int alto;
    /* Lista de paneles contenidos */
    std::list <PanelContenedor *> _paneles;
    /* Corrige las coordenadas */
    void corregirCoordenadas(ulong *x, ulong *y);

public:
    /* Constructor y destructor */
    Ventana(int ancho, int alto);
    virtual ~Ventana();
    /* Recibe eventos y dibuja */
    void ejecutar ();
    /* Dibuja todos los paneles, Reloje y Plata */
    void imprimir_pantalla (void);
    /* Agrega paneles */
    void agregarPanel(PanelContenedor *, const Coordenada);
    void agregarPanel(PanelContenedor * panel, const int lugar);
    /* Ejecuta eventos de paneles o propios */
    bool seEjecutaEventoEn(const Coordenada, int evento);
    /* Levanta a los botones porque otro se clickeo */
    void desclickearBotones(int id_excepcion);

    /* Eventos del mouse */
    void evento_MouseDown(Coordenada punto);
    void evento_MouseOver(Coordenada punto);
    void evento_MouseUp(Coordenada punto);
    /* variables utilitarias */
    bool salir;
    bool sms;

};

#endif /*VENTANA_H_*/
```

```
#ifndef __VENTANA_CONFIGURACION_H__
#define __VENTANA_CONFIGURACION_H__


#include "Mapa.h"
#include <glib.h>
#include <glib/gprintf.h>
#include <gtk/gtk.h>
#include <string>

class VentanaConfiguracion;

/* Estructura de datos auxiliar para pasar parámetros */
typedef struct datosConfiguracion
{
    GtkWidget* textoNom;
    GtkWidget* textoPlata;
    VentanaConfiguracion* pthis;
    GtkWidget* ventana;

    //Botones de elección
    GtkWidget* elMasa;
    GtkWidget* elPuntoFijo;
    GtkWidget* elRueda;
    GtkWidget* elBarra;
    GtkWidget* elPlataforma;
    GtkWidget* elSoga;
    GtkWidget* elCinta;
    GtkWidget* elCohete;

    //Espacios de entrada
    GtkWidget* textoPlataMasa;
    GtkWidget* textoPlataPuntoFijo;
    GtkWidget* textoPlataRueda;
    GtkWidget* textoPlataBarra;
    GtkWidget* textoPlataPlataforma;
    GtkWidget* textoPlataSoga;
    GtkWidget* textoPlataCinta;
    GtkWidget* textoPlataCohete;
    GtkWidget* textoTiempoCohete;

}datosConfiguracion;

class VentanaConfiguracion
{
private:
    Mapa* mapa; //Mapa editado
    datosConfiguracion* datos;

    /* Cierra una ventana pasada por parámetro */
    static void cerrar_ventana(GtkWidget *widget, gpointer data);

    /* Analiza si la cadena es un texto vacío o únicamente de espacios */
    static bool textoVacio(const std::string& texto);

    /* Muestra un GtkDialog con la leyenda dada */
    static void mostrarAlerta(std::string& leyenda, GtkWidget* ventanaPadre);

    /* Cierra Gtk */
    static void destruir(GtkWidget *widget, gpointer data);

    /* Pasa de entero a string */
    static void enteroAString(std::string& cadena, ulong entero);

    /* Analiza si la cadena está formada por números o no */
    static bool esNumero(const std::string& cadena);

public:
    /* Constructor */
    VentanaConfiguracion(Mapa* mapa, GtkWidget* ventanaPpal);
```

```
/* Destructor */
virtual ~VentanaConfiguracion();

/* Persiste el mapa que se está editando */
static void guardar(GtkWidget *widget, gpointer data);

};

#endif
```

```
#ifndef VIDEO_H
#define VIDEO_H

#include "config.h"

#include "loader.h"
#include <cmath>
#include <fstream>
#include <SDL.h>
#include <SDL_opengl.h>
#include <sstream>
#include <iostream>
#include <vector>

class Video
{
public:
    Video(const Config& config);
    virtual ~Video();
    GLuint texture;           // This is a handle to our texture object
    SDL_Surface *surface;    // This surface will tell us the details of the image
    GLenum texture_format;
    GLint nOfColors;
    std::vector<GLuint> textures;
    void loadSurfaces();
    void loadSurface(char* surf);
};

#endif
```

```
#ifndef ZONA_LLEGADA_H
#define ZONA_LLEGADA_H

class ZonaLlegada
{
public:
    ZonaLlegada(double x, double y, double width, double height);
    double getX();
    double getY();
    double getWidth();
    double getHeight();
private:
    double x;
    double y;
    double width;
    double height;

};

#endif
```

```
#include "Angulo.h"
#include <math.h>
#include <iostream>
using namespace std;

// hay q hacer bien esta funcion
void Angulo::corregir(long* valor)
{
    long copia = *valor;

    if(fabs(*valor) >= 360.0)
    {
        long vueltas = copia / 360.0;
        copia = copia - (vueltas*360.0);
    }

    if(*valor < 0)
        *valor = 360.0 - copia;
    else
        *valor = copia;
}

double Angulo::aRadianes(long grados) {
    return (2.0*PI * (double)grados) / 360.0;
}

long Angulo::aGrados(double radianes) {
    return ((radianes * 360.0) / (2.0*PI));
}
```

```
#include "Barra.h"

using namespace std;

Barra::Barra() :
    ElementoLongitudinal("Longitudinal:Barra", Coordenada(long(10), long(10)), 25, 0, 10) {
}

Barra::Barra(Cordenada posicion, ulong magnitud, long angulo, long ancho) :
    ElementoLongitudinal("Longitudinal:Barra", posicion, magnitud, angulo, ancho) {
}

Barra::~Barra() {
```

```
#include "Boton.h"
#include <iostream>
#include "constantes.h"
using namespace std;

Boton::Boton(int nroDeBoton, Mapa * mapa) : PanelContenedor("Boton", getRutaImgDefecto(nroDeBoton),
getRutaImgMouseDown(nroDeBoton), getRutaImgMouseOver(nroDeBoton)) {
    pf_rutinaMouseDown = NULL;
    pf_rutinaMouseOver = NULL;
    pf_rutinaMouseUp = NULL;
    this->mapa = mapa;
    presionado = false;
}

Boton::~Boton() {}

bool Boton::esta_presionado() const {
    return presionado;
}

void Boton::desapretar() {
    presionado = false;
}

void Boton::evento_MouseDown(Coordenada punto) {
    PanelContenedor::evento_MouseDown(punto);
    ventana->desclickearBotones(this->getId());
    if(presionado == false)
        presionado = true;
    else
        presionado = false;
    if(pf_rutinaMouseDown != NULL)
        pf_rutinaMouseDown(punto, this);
}

void Boton::evento_MouseUp(Coordenada punto) {
    PanelContenedor::evento_MouseUp(punto);

    if(pf_rutinaMouseUp != NULL)
        pf_rutinaMouseUp(punto, this);
}

void Boton::evento_MouseOver(Coordenada punto) {
    PanelContenedor::evento_MouseOver(punto);
    if(pf_rutinaMouseOver != NULL)
        pf_rutinaMouseOver(punto, this);
}

void Boton::setRutinaMouseDown(void (*rutina)(Coordenada, Boton *)) {
    pf_rutinaMouseDown = rutina;
}

void Boton::setRutinaMouseUp(void (*rutina)(Coordenada, Boton *)) {
    pf_rutinaMouseUp = rutina;
}

void Boton::setRutinaMouseOver(void (*rutina)(Coordenada, Boton *)) {
    pf_rutinaMouseOver = rutina;
}

Mapa * Boton::getMapa() const {
    return mapa;
}

void Boton::enviarSenialSalida() {
    ventana->salir = true;
}

void Boton::enviarSenialSms() {
```

```
if(!ventana->sms)
    ventana->sms = true;
else
    ventana->sms = false;
}

bool Boton::getSmsState(){
    return ventana->sms;
}

string Boton::getRutaImgDefecto(int nroDeBoton) {
    switch(nroDeBoton) {
        case MASA: return "./ima/botones/icon_level_off.png";
        case RUEDA: return "./ima/botones/PlataformaMetalica_Defecto.bmp";
        case PUNTO_FIJO: return "./ima/botones/icon_node_off.png";
        case BARRA: return "./ima/botones/icon_metal_bar_off.png";
        case PLATAFORMA: return "./ima/botones/icon_metal_sheet_off.png";
        case SOGA: return "./ima/botones/icon_elastic_off.png";
        case CINTA: return "./ima/botones/icon_cloth_off.png";
        case COHETE: return "./ima/botones/icon_rocket_off.png";
        case BORRAR_TODO: return "./ima/botones/icon_clear_off.png";
        case BORRAR_ELEMENTO: return "./ima/botones/icon_rubber_off.png";
        case PLAY: return "./ima/botones/icon_start_off.png";
        case SALIR: return "./ima/botones/icon_salir_off.png";
        case SMS: return "./ima/botones/icon_sms_off.png";
        case ZONA_LLEGADA: return "./ima/botones/icon_llegada_off.png";
        case GUARDAR: return "./ima/botones/icon_save_off.png";
        case VOID: return "./letras/vacio.bmp";
    }
    return "";
}

string Boton::getRutaImgMouseDown(int nroDeBoton) {
    switch(nroDeBoton) {
        case MASA: return "./ima/botones/icon_level_on.png";
        case RUEDA: return "./ima/botones/PlataformaMetalica_MouseDown.bmp";
        case PUNTO_FIJO: return "./ima/botones/icon_node_on.png";
        case BARRA: return "./ima/botones/icon_metal_bar_on.png";
        case PLATAFORMA: return "./ima/botones/icon_metal_sheet_on.png";
        case SOGA: return "./ima/botones/icon_elastic_on.png";
        case CINTA: return "./ima/botones/icon_cloth_on.png";
        case COHETE: return "./ima/botones/icon_rocket_on.png";
        case BORRAR_TODO: return "./ima/botones/icon_clear_on.png";
        case BORRAR_ELEMENTO: return "./ima/botones/icon_rubber_on.png";
        case PLAY: return "./ima/botones/icon_start_on.png";
        case SALIR: return "./ima/botones/icon_salir_on.png";
        case SMS: return "./ima/botones/icon_sms_on.png";
        case GUARDAR: return "./ima/botones/icon_save_on.png";
        case ZONA_LLEGADA: return "./ima/botones/icon_llegada_on.png";
        case VOID: return "./letras/vacio.bmp";
    }
    return "";
}

string Boton::getRutaImgMouseOver(int nroDeBoton) {
    switch(nroDeBoton) {
        case MASA: return "./ima/botones/icon_level_over.png";
        case RUEDA: return "./ima/botones/PlataformaMetalica_MouseOver.bmp";
        case PUNTO_FIJO: return "./ima/botones/icon_node_over.png";
        case BARRA: return "./ima/botones/icon_metal_bar_over.png";
        case PLATAFORMA: return "./ima/botones/icon_metal_sheet_over.png";
        case SOGA: return "./ima/botones/icon_elastic_over.png";
        case CINTA: return "./ima/botones/icon_cloth_over.png";
        case COHETE: return "./ima/botones/icon_rocket_over.png";
        case BORRAR_TODO: return "./ima/botones/icon_clear_over.png";
        case BORRAR_ELEMENTO: return "./ima/botones/icon_rubber_over.png";
        case PLAY: return "./ima/botones/icon_start_over.png";
        case SALIR: return "./ima/botones/icon_salir_over.png";
        case SMS: return "./ima/botones/icon_sms_over.png";
    }
}
```

```
        case GUARDAR: return "./ima/botoness/icon_save_over.png";
        case ZONA_LLEGADA: return "./ima/botoness/icon_llegada_over.png";
        case VOID: return "./letras/vacio.bmp";
    }
    return "";
}
```

```
#include "Cinta.h"

Cinta::Cinta() :
    ElementoLongitudinal("Longitudinal:Cinta", Coordenada(long(10), long(10)), 25, 0, 10) {}

Cinta::Cinta(Cordenada posicion, ulong magnitud, long angulo, long ancho) :
    ElementoLongitudinal("Longitudinal:Cinta", posicion, magnitud, angulo,
                           ancho) {}

Cinta::~Cinta() {}
```

```
#include "Cohete.h"

Cohete::Cohete() :
    ElementoLongitudinal("Longitudinal:Cohete", Coordenada(long(10), long(10)), 25, 0, 10) {}

Cohete::Cohete(Cordenada posicion, ulong magnitud, long angulo, long ancho) :
    ElementoLongitudinal("Longitudinal:Cohete", posicion, magnitud, angulo,
                           ancho) {}

Cohete::~Cohete() {
```

```
#include "common_Mutex.h"

Mutex::Mutex()
{
    pthread_mutex_init(&mut, NULL);
}

void Mutex::lock()
{
    pthread_mutex_lock(&mut);
}

void Mutex::unlock()
{
    pthread_mutex_unlock(&mut);
}
```

```
#include "common_Thread.h"

#include <signal.h>
#include <unistd.h>

Thread::Thread()
{
    vivo = true;
}

void Thread::start()
{
    corriendo = true;
    pthread_create(&hilo,NULL,static_run,this);
}

Thread::~Thread()
{
    if (corriendo)
    {
        vivo = false; //Indica al hilo que debe suicidarse.
        this->join();
    }
}

void Thread::join() const
{
    pthread_join(hilo,NULL);
}

bool Thread::estaVivo()
{
    bool retorno;
    bloquearEstado();
    retorno = vivo;
    desbloquearEstado();
    return retorno;
}

void* Thread::static_run (void* p)
{
    Thread* pthis = (Thread*) p;
    pthis->run();
    return NULL;
}

void Thread::terminar()
{
    bloquearEstado();
    vivo = false;
    desbloquearEstado();
}

bool Thread::estaCorriendo()
{
    bool retorno;
    bloquearEstado();
    retorno = corriendo;
    desbloquearEstado();
    return retorno;
}

void Thread::bloquearEstado()
```

```
mHilo.lock();  
}  
  
void Thread::desbloquearEstado()  
{  
    mHilo.unlock();  
}  
  
void Thread::setCorriendo(bool valor)  
{  
    corriendo = valor;  
}
```

```
#include "config_file.h"
#include <fstream>

namespace
{
    void readConfigData(std::map<std::string, std::string>& data,
                        std::ifstream& configFile)
    {
        std::string line;

        while (std::getline(configFile, line))
        {
            if (line[0] == '#')
                continue;

            size_t eqPos = line.find('=');
            if (eqPos == linenpos)
                continue;
            std::string key = line.substr(0, eqPos);
            std::string value = line.substr(eqPos + 1);

            data[key] = value;
        }
    }

ConfigFile::ConfigFile(const std::string& filename)
{
    std::ifstream configFile(filename.c_str());

    if (configFile.is_open())
        readConfigData(data_, configFile);
}

ConfigFile::~ConfigFile()
{
}

std::string ConfigFile::readString(const std::string& key) const
{
    TStrStrMap::const_iterator it = data_.find(key);
    if (it == data_.end())
        return std::string();
    else
        return it->second;
}
```

```
#include "Coordenada.h"
#include <math.h>
#include <iostream>
#include "constantes.h"

using namespace std;

Coordenada::Coordenada() {

}

Coordenada::Coordenada(long x, long y) {
    this->x = x;
    this->y = y;
    actualizarAngulo();
    actualizarModulo();
}

Coordenada::Coordenada(ulong modulo, long angulo) {
    this->modulo = modulo;
    this->angulo = angulo;
    actualizarX();
    actualizarY();
}

Coordenada::~Coordenada() {
}

void Coordenada::modificarDelta(Cordenada delta) {
    x += delta.x;
    y += delta.y;
}

void Coordenada::modificar(Cordenada nueva) {
    x = nueva.x;
    y = nueva.y;
}

void Coordenada::imprimir() const {
    cout << "< " << this->x << " , " << this->y << " > " << endl;
    /*
     << "Angulo: "
         << angulo
         << " Modulo: "
         << modulo << endl;
    */
}

void Coordenada::setX(long x) {
    this->x = x;
    actualizarAngulo();
    actualizarModulo();
}

void Coordenada::setY(long y) {
    this->y = y;
    actualizarAngulo();
    actualizarModulo();
}

void Coordenada::setAngulo(long angulo) {
    this->angulo = angulo;
    actualizarX();
    actualizarY();
}

void Coordenada::setModulo(ulong modulo) {
    this->modulo = modulo;
    actualizarX();
    actualizarY();
}
```

```
Coordenada Coordenada::operator+(const Coordenada c) const {
    Coordenada aux = Coordenada(this->x + c.x, this->y + c.y);
    return aux;
}

Coordenada Coordenada::operator-(const Coordenada c) const {
    Coordenada aux = Coordenada(this->x - c.x, this->y - c.y);
    return aux;
}

void Coordenada::actualizarModulo() {
    modulo = sqrt(pow(x, 2) + pow(y, 2));
}

void Coordenada::actualizarAngulo() {
    angulo = Angulo::aGrados(atan2(y,x));
    Angulo::corregir(&angulo);
}

void Coordenada::actualizarX() {
    this->x = modulo * cos(Angulo::aRadianes(angulo));
}

void Coordenada::actualizarY() {
    y = modulo * sin(Angulo::aRadianes(angulo));
}
```

```
//cronometro.cpp

#include "cronometro.h"
int Cronometro::estaParado() { return parado; }

void Cronometro::iniciar() {
    comienzo=clock();
    parado=0;
}

void Cronometro::detener() {
    final=clock();
    parado=1;
}

long Cronometro::getMilisegundos() {
    if (estaParado()) return ((final-comienzo)*1000.0)/CLOCKS_PER_SEC;
    return ((clock()-comienzo)*1000.0)/CLOCKS_PER_SEC;
}
```

```
#include "element.h"

Element::Element(unsigned f, unsigned l){
    first = f;
    last = l;
}

unsigned Element::getFirst(){
    return masas.at(0);
}

unsigned Element::getLast(){
    return masas.at(masas.size()-1);
}
```

```
#include "elementManager.h"

ElementManager::ElementManager(){
    vectoresElementos.push_back(&ropes);
    vectoresElementos.push_back(&cohetes);
    vectoresElementos.push_back(&ruedas);
    vectoresElementos.push_back(&plataformas);
    vectoresElementos.push_back(&metalBars);
    vectoresElementos.push_back(&mainBalls);
    vectoresElementos.push_back(&lonas);

}

ElementManager::~ElementManager(){
    for (size_t i=0; i< vectoresElementos.size(); i++)
        for (size_t j=0; j< vectoresElementos.at(i)->size(); j++)
            delete(vectoresElementos.at(i)->at(j));

    for (size_t i=0; i< zonasLlegada.size(); i++)
        delete(zonasLlegada.at(i));
}

void ElementManager::addRope(int first, int last){
    Element* elemento = new Element(first,last);
    for(int i=first; i<=last;i++)
        elemento->masas.push_back(i);

    ropes.push_back(elemento);
}

void ElementManager::addCohete(int first, int last){
    Element* elemento = new Element(first,last);
    for(int i=first; i<=last;i++)
        elemento->masas.push_back(i);

    cohetes.push_back(elemento);
}

void ElementManager::addRueda(int first, int last){
    Element* elemento = new Element(first,last);
    for(int i=first; i<=last;i++)
        elemento->masas.push_back(i);

    ruedas.push_back(elemento);
}

void ElementManager::addMetalBar(int first, int last){
    Element* elemento = new Element(first,last);
    for(int i=first; i<=last;i++)
        elemento->masas.push_back(i);

    metalBars.push_back(elemento);
}

void ElementManager::addPlataforma(int first, int last){
    Element* elemento = new Element(first,last);
    for(int i=first; i<=last;i++)
        elemento->masas.push_back(i);

    plataformas.push_back(elemento);
}

void ElementManager::addMainBall(int first, int last){
    Element* elemento = new Element(first,last);
    for(int i=first; i<=last;i++)
        elemento->masas.push_back(i);

    mainBalls.push_back(elemento);
}

void ElementManager::addLona(int first, int last){
```

```
Element* elemento = new Element(first,last);
for(int i=first; i<=last;i++)
    elemento->masas.push_back(i);

lonas.push_back(elemento);
}

void ElementManager::addPuntoFijo(int first, int last){
    Element* elemento = new Element(first,last);
    elemento->masas.push_back(first);
    puntosFijos.push_back(elemento);
}

void ElementManager::addZonaLlegada(double x,double y,double width,double heigth){
    zonasLlegada.push_back(new ZonaLlegada(x,y,width,heigth));
}

void ElementManager::addSpringElement(int spring, int element){
    springElements.insert(std::make_pair(spring,element));
}

std::vector<Element*> ElementManager::getRopes(){
    return ropes;
}
std::vector<Element*> ElementManager::getPuntosFijos(){
    return puntosFijos;
}
std::vector<Element*> ElementManager::getRuedas(){
    return ruedas;
}
std::vector<Element*> ElementManager::getMainBalls(){
    return mainBalls;
}
std::vector<Element*> ElementManager::getLonas(){
    return lonas;
}
std::vector<Element*> ElementManager::getCohetes(){
    return cohetes;
}
std::vector<Element*> ElementManager::getPlataformas(){
    return plataformas;
}
std::vector<Element*> ElementManager::getMetalBars(){
    return metalBars;
}
std::vector<ZonaLlegada*> ElementManager::getZonasLlegada(){
    return zonasLlegada;
}
std::map<int,int> ElementManager::getSpringElements(){
    return springElements;
}
```

```
#include "Elemento.h"
#include "Angulo.h"
#include "math.h"

using namespace std;

Elemento::Elemento() {
    this->bloqueado = false;
}

Elemento::Elemento(const std::string & nombre, Coordenada posicion,
                   const ulong magnitud, long angulo) {
    this->nombre = nombre;
    this->posicion = posicion;
    this->magnitud = magnitud;
    Angulo::corregir(&angulo);
    this->angulo = angulo;
    this->bloqueado = false;
    this->mostrarEnlaces = false;
}

Elemento::~Elemento() {}

void Elemento::setMagnitud(const long valor) {
    if(!bloqueado)
        this->magnitud = valor;
}

ulong Elemento::getMagnitud() const {
    return this->magnitud;
}

void Elemento::setPosicion(const Coordenada nuevaPosicion) {
    if(!bloqueado)
        this->posicion = nuevaPosicion;
}

Coordenada Elemento::getPosicion() const {
    return this->posicion;
}

void Elemento::setAngulo(const long angulo) {
    if(!bloqueado)
        this->angulo = angulo;
}

long Elemento::getAngulo() const {
    return this->angulo;
}

Coordenada Elemento::getExtremoOpuesto() const {
    Coordenada extOp;
    extOp.setX(magnitud * cos(Angulo::aRadianes(angulo)) + posicion.x);
    extOp.setY(magnitud * sin(Angulo::aRadianes(angulo)) + posicion.y);
    return extOp;
}

void Elemento::bloquear() {
    bloqueado = true;
}

bool Elemento::estaBloqueado() const {
    return bloqueado;
}

string Elemento::getNombre() const {
    return nombre;
}

bool Elemento::es(const int nro_elemento) const {
    switch(nro_elemento) {
```

```
case MASA: return (nombre.find("Masa")!=string::npos)? true : false;
case PUNTO_FIJO: return (nombre.find("PuntoFijo")!=string::npos)? true : false;
case RUEDA: return (nombre.find("Rueda")!=string::npos)? true : false;
case BARRA: return (nombre.find("Barra")!=string::npos)? true : false;
case PLATAFORMA: return (nombre.find("Plataforma")!=string::npos)? true : false;
case CINTA: return (nombre.find("Cinta")!=string::npos)? true : false;
case SOGA: return (nombre.find("Soga")!=string::npos)? true : false;
case COHETE: return (nombre.find("Cohete")!=string::npos)? true : false;
case CIRCULAR: return (nombre.find("Circular")!=string::npos)? true : false;
case LONGITUDINAL: return (nombre.find("Longitudinal")!=string::npos)? true : false;
case ENLAZABLE: return (nombre.find("Enlazable")!=string::npos)? true : false;
case PUNTO_ENLACE: return (nombre.find("PuntoDeEnlace")!=string::npos)? true : false;
case ZONA_LLLEGADA: return (nombre.find("ZonaLlegada")!=string::npos)? true : false;
}
return false;
}

int Elemento::getTipo() const
{
    string nom = nombre.substr(nombre.find_last_of(":")+1,nombre.length()-nombre.find_last_of(":")-1);
    if (nom.compare("Masa") == 0)
        return MASA;

    if (nom.compare("Rueda") == 0)
        return RUEDA;

    if (nom.compare("PuntoFijo") == 0)
        return PUNTO_FIJO;

    if (nom.compare("Barra") == 0)
        return BARRA;

    if (nom.compare("Plataforma") == 0)
        return PLATAFORMA;

    if (nom.compare("Soga") == 0)
        return SOGA;

    if (nom.compare("Cinta") == 0)
        return CINTA;

    if (nom.compare("Cohete") == 0)
        return COHETE;

    if (nom.compare("ZonaLlegada") == 0)
        return ZONA_LLLEGADA;

    return NINGUN_ELEMENTO;
}

void Elemento::imprimir() {
    cout << "Elemento: " << getNombre() << endl;
    cout << "Posicion actual: ";
    this->posicion.imprimir();
    cout << "Magnitud: " << this->magnitud << endl
        << "Angulo actual: " << this->angulo << endl << endl;
}

bool Elemento::getMostrarPuntosDeEnlace() const {
    return mostrarEnlaces;
}

void Elemento::setMostrarPuntosDeEnlace(const bool b) {
    mostrarEnlaces = b;
}
```

```
#include "ElementoCircular.h"

ElementoCircular::ElementoCircular(const std::string & nombre,
                                    Coordenada posicion, const ulong magnitud, long angulo) :
    Elemento(nombre, posicion, magnitud, angulo) {

}

ElementoCircular::ElementoCircular() {

}

ElementoCircular::~ElementoCircular() {

}

int ElementoCircular::seleccionar(Coordenada punto) const {
    int seleccion= ELEMENTO_NO_SELECCIONADO;
    Coordenada resta = punto - getPosition();

    if (resta.modulo <= getMagnitud()) {
        seleccion = ELEMENTO_SELECCIONADO POR CENTRO;
    }

    return seleccion;
}
```

```
#include "ElementoEnlazable.h"

using namespace std;

ElementoEnlazable::ElementoEnlazable(const std::string & nombre,
                                      Coordenada posicion, const ulong magnitud, long angulo, Elemento * poseedor) :
    ElementoCircular(nombre, posicion, magnitud, angulo) {
    this->poseedor = poseedor;
    enlazado = false;
}

ElementoEnlazable::~ElementoEnlazable() {
}

Elemento * ElementoEnlazable::getPoseedor() const {
    return poseedor;
}

list <ElementoEnlazable*> ElementoEnlazable::getListaEnlaces() const {
    return _enlaces;
}

void ElementoEnlazable::eliminarEnlaces() {
    list <ElementoEnlazable*>::iterator it;
    for(it = _enlaces.begin(); it != _enlaces.end(); it++) {
        delete (*it);
    }
    _enlaces.clear();
}

bool ElementoEnlazable::conectarA(ElementoEnlazable* enganche) {
    if(enganche != this)
        if((this->getPosicion().x == enganche->getPosicion().x) && (this->getPosicion().y ==
enganche->getPosicion().y)) {
            _enlaces.push_back(enganche);
            return true;
        }
    return false;
}
```

```
#include "ElementoLongitudinal.h"
#include <math.h>
#include <iostream>
using namespace std;

ElementoLongitudinal::ElementoLongitudinal() {

}

ElementoLongitudinal::ElementoLongitudinal(const std::string & nombre,
                                             Coordenada posicion, const ulong magnitud, long angulo, ulong ancho) :
    Elemento(nombre, posicion, magnitud, angulo) {
    Coordenada p2;
    p2.setAngulo(angulo);
    p2.setModulo(magnitud);
    p2 = p2 + posicion;
    extremoOpuesto = p2;
    this->ancho = ancho;
}

ElementoLongitudinal::~ElementoLongitudinal() {

}

void ElementoLongitudinal::setPosicion(const Coordenada nuevaPosicion) {
    Elemento::setPosicion(nuevaPosicion);

    Coordenada p2;
    long ang = -getAngulo();

    p2.setX( cos(Angulo::aRadianes(ang))*getMagnitud() + getPosicion().x );
    p2.setY( sin(Angulo::aRadianes(ang))*getMagnitud() + getPosicion().y );
    extremoOpuesto = p2;
}

void ElementoLongitudinal::setAngulo(const long angulo) {
    Elemento::setAngulo(angulo);
}

void ElementoLongitudinal::actualizarPuntosDeEnlace(int modo) {
    int cantidadEnlaces = getMagnitud() / LONGITUD_DE_ENLACE_MINIMA;
    if(modo == 1) cantidadEnlaces = 1;

    int separacionEntreEnlaces = (cantidadEnlaces == 0) ? this->getMagnitud()
                                                       : (getMagnitud() / cantidadEnlaces);

    eliminarEnlaces();
    _enlaces.clear();

    /* Completa la lista de coordenadas de enlace sobre el elemento longitudinal */
    for(ulong paso = 0; paso <= (this->getMagnitud() - separacionEntreEnlaces); paso += separacionEntreEnlaces) {
        Coordenada posicionEnlace;
        posicionEnlace.setModulo(paso);
        posicionEnlace.setAngulo(-this->getAngulo());
        posicionEnlace = posicionEnlace + this->getPosicion();
        PuntoDeEnlace * nuevoEnlace = new PuntoDeEnlace(posicionEnlace, this);
        _enlaces.push_back(nuevoEnlace);
    }
    _enlaces.push_back(new PuntoDeEnlace(this->getExtremoOpuesto(), this));
}

void ElementoLongitudinal::eliminarEnlaces() {
    list <PuntoDeEnlace*>::iterator it;
    for(it = _enlaces.begin(); it != _enlaces.end(); it++) {
        delete (*it);
    }
    _enlaces.clear();
}
```

```
void ElementoLongitudinal::imprimirEnlaces() {
    list <PuntoDeEnlace*>::iterator it;
    for(it = _enlaces.begin(); it != _enlaces.end(); it++) {
        (*it)->getPosicion().imprimir();
    }
    cout << endl << endl;
}

std::list<PuntoDeEnlace*> ElementoLongitudinal::getListaEnlaces() {
    return _enlaces;
}

ulong ElementoLongitudinal::getAncho() const {
    return ancho;
}

void ElementoLongitudinal::setExtremo0puesto(const Coordenada p2) {
    extremo0puesto = p2;

    Coordenada resta = p2 - getPosicion();
    setMagnitud(resta.modulo);

    if (resta.angulo<180)
        resta.angulo=-resta.angulo;

    setAngulo(resta.angulo);
}

Coordenada ElementoLongitudinal::getExtremo0puesto() const {
    return extremo0puesto;
}

int ElementoLongitudinal::seleccionar(Cordenada punto) const {
    Coordenada aux = punto - this->getPosicion(); //traslado el punto al origen

    if (aux.angulo<180)
        aux.angulo=-aux.angulo;

    aux.setAngulo(aux.angulo - this->getAngulo()); //roto

    int resultado= ELEMENTO_NO_SELECCIONADO;

    /* Analisis de si se hizo click y en donde */
    if ( (aux.x <= this->getMagnitud()) && (fabs(aux.y) <= (this->ancho +
MARGEN_DE_AGARRE_ANCHO)) ) {
        resultado = ELEMENTO_SELECCIONADO_POR_CENTRO;
        if (aux.x > (long)(this->getMagnitud() - MARGEN_DE_AGARRE))
            resultado = ELEMENTO_SELECCIONADO_POR_EXTRÉM02;
    }

    return resultado;
}
```

```
#include "FabricaDeElementos.h"

FabricaDeElementos::FabricaDeElementos() {
}

FabricaDeElementos::~FabricaDeElementos() {
}

Elemento * FabricaDeElementos::crear(int nroElemento, Coordenada posicion) {
    switch (nroElemento) {
        case MASA:
            return new Masa(posicion, 20);
        case RUEDA:
            return new Rueda(posicion, 30);
        case PUNTO_FIJO:
            return new PuntoFijo(posicion, 25);
        case BARRA:
            return new Barra(posicion, LONGITUD_MINIMA, 0, 5);
        case PLATAFORMA:
            return new Plataforma(posicion, LONGITUD_MINIMA, 0, 6);
        case SOGA:
            return new Soga(posicion, LONGITUD_MINIMA, 0, 5);
        case CINTA:
            return new Cinta(posicion, LONGITUD_MINIMA, 0, 5);
        case COHETE:
            return new Cohete(posicion, LONGITUD_COHETE , 0, 20);
        case ZONA_LLEGADA:
            return new Llegada(posicion);
    }
    return NULL;
}
```

```
#include "fileutils.h"
#include <SDL.h>

BinFile::BinFile(const char *filename) :
s_(filename, std::ios::in | std::ios::binary)
{
}

BinFile::~BinFile()
{
}

char BinFile::readbyte()
{
    return s_.get();
}

int16_t BinFile::readword()
{
    uint16_t ret;
    s_.read(reinterpret_cast<char*>(&ret), sizeof(ret));
#ifndef SDL_BYTEORDER != SDL_LIL_ENDIAN
    ret = SDL_Swap16(ret);
#endif
    return ret;
}

int32_t BinFile::read dword()
{
    uint32_t ret;
    s_.read(reinterpret_cast<char*>(&ret), sizeof(ret));
#ifndef SDL_BYTEORDER != SDL_LIL_ENDIAN
    ret = SDL_Swap32(ret);
#endif
    return ret;
}

void BinFile::readbytes(char* buffer, size_t num)
{
    s_.read(buffer, num);
}

size_t BinFile::tell()
{
    return s_.tellg();
}

void BinFile::seek(size_t pos)
{
    s_.seekg(pos);
}
```

```
#include "hiloCron.h"
using namespace std;

HiloCron::HiloCron()
{
}

HiloCron::~HiloCron()
{
}

void HiloCron::run()
{
    cronometro = new Cronometro();
    cronometro->iniciar();
    while (estaVivo())
    {
        sleep(0);
    }
    cronometro->detener();
    delete(cronometro);
    cronometro = NULL;
}

long HiloCron::getMilisegundos(){
    if (cronometro) return cronometro->getMilisegundos();
    return -1;
}
```

```
#include "InterfazEdicion.h"
#include "sms_app.h"

#include "ParserXML.h"

using namespace std;

InterfazEdicion::InterfazEdicion(Mapa* mapa) {
    this->mapa = mapa;
}

InterfazEdicion::~InterfazEdicion() {
}

void InterfazEdicion::comenzar() {
    Ventana miVentana = Ventana(ANCHO_VENTANA, ALTO_VENTANA);
    agregarBotoneraInferior(&miVentana);
    string rutaFondo = "./ima/mapa/fondoBasico.bmp";
    InterfazMapa * interfazMapa = new InterfazMapa(rutaFondo, rutaFondo, mapa);
    interfazMapa->setAltura(mapa->getAlto());
    interfazMapa->setAncho(mapa->getAncho());
    interfazMapa->setRutinaMouseDown(mapa_mouseDown);
    interfazMapa->setRutinaMouseOver(mapa_mouseOver);
    interfazMapa->setRutinaMouseUp(mapa_mouseUp);
    miVentana.agregarPanel(interfazMapa, Coordenada((long)0, (long)0));
    miVentana.ejecutar();
}

void InterfazEdicion::agregarBotoneraInferior(Ventana * miVentana) {
    /* Boton Masa */
    Boton * botonMasa = new Boton(MASA, mapa);
    botonMasa->setRutinaMouseDown(botonMasa_mouseDown);
    miVentana->agregarPanel(botonMasa, 0);

    /* Boton Rueda */
    Boton * botonRueda = new Boton(RUEDA, mapa);
    botonRueda->setRutinaMouseDown(botonRueda_mouseDown);
    miVentana->agregarPanel(botonRueda, 0);

    /* Boton PuntoFijo */
    Boton * botonPuntoFijo = new Boton(PUNTO_FIJO, mapa);
    botonPuntoFijo->setRutinaMouseDown(botonPuntoFijo_mouseDown);
    miVentana->agregarPanel(botonPuntoFijo, 0);

    /* Boton Barra */
    Boton * botonBarra = new Boton(BARRA, mapa);
    botonBarra->setRutinaMouseDown(botonBarra_mouseDown);
    miVentana->agregarPanel(botonBarra, 0);

    /* Boton PLataforma */
    Boton * botonPLataforma = new Boton(PLATAFORMA, mapa);
    botonPLataforma->setRutinaMouseDown(botonPlataforma_mouseDown);
    miVentana->agregarPanel(botonPLataforma, 0);

    /* Boton Soga */
    Boton * botonSoga = new Boton(SOGA, mapa);
    botonSoga->setRutinaMouseDown(botonSoga_mouseDown);
    miVentana->agregarPanel(botonSoga, 0);

    /* Boton Cinta */
    Boton * botonCinta = new Boton(CINTA, mapa);
    botonCinta->setRutinaMouseDown(botonCinta_mouseDown);
```

```
miVentana->agregarPanel(botonCinta, 0);

/* Boton ZonaLlegada */
Boton * botonZonaLlegada = new Boton(ZONA_LLEGADA, mapa);
botonZonaLlegada->setRutinaMouseDown(botonZonaLlegada_mouseDown);
miVentana->agregarPanel(botonZonaLlegada, 0);

/* Boton Cohete */
Boton * botonCohete = new Boton(COHETE, mapa);
botonCohete->setRutinaMouseDown(botonCohete_mouseDown);
miVentana->agregarPanel(botonCohete, 0);

/* Boton Borrar Elemento */
Boton * botonBorrarElemento = new Boton(BORRAR_ELEMENTO, mapa);
botonBorrarElemento->setRutinaMouseDown(botonBorrarElemento_mouseDown);
botonBorrarElemento->setAltura(32);
botonBorrarElemento->setAncho(32);
miVentana->agregarPanel(botonBorrarElemento, 0);

/* Boton Borrar Todo */
Boton * botonBorrarTodo = new Boton(BORRAR_TODO, mapa);
botonBorrarTodo->setRutinaMouseDown(botonBorrarTodo_mouseDown);
botonBorrarTodo->setAltura(32);
botonBorrarTodo->setAncho(32);
miVentana->agregarPanel(botonBorrarTodo, 0);

/* Boton Void */
Boton * botonVoid = new Boton(VOID, mapa);
botonVoid->setRutinaMouseDown(NULL);
botonVoid->setAltura(32);
botonVoid->setAncho(32);
miVentana->agregarPanel(botonVoid, 0);

/* Boton Guardar */
Boton * botonGuardar = new Boton(GUARDAR, mapa);
botonGuardar->setRutinaMouseDown(botonGuardar_mouseDown);
botonGuardar->setAltura(32);
botonGuardar->setAncho(32);
miVentana->agregarPanel(botonGuardar, 0);

/* Boton PLAY */
Boton * botonPlay = new Boton(PLAY, mapa);
botonPlay->setRutinaMouseDown(botonPlay_mouseDown);
botonPlay->setAltura(32);
botonPlay->setAncho(32);
miVentana->agregarPanel(botonPlay, 0);

/* Boton SMS */
Boton * botonSms = new Boton(SMS, mapa);
botonSms->setRutinaMouseDown(botonSms_mouseDown);
botonSms->setAltura(32);
botonSms->setAncho(32);
miVentana->agregarPanel(botonSms, 0);

/* Boton Salir */
Boton * botonSalir = new Boton(SALIR, mapa);
botonSalir->setRutinaMouseDown(botonSalir_mouseDown);
botonSalir->setAltura(32);
botonSalir->setAncho(32);
miVentana->agregarPanel(botonSalir, 0);

}

/* EVENTOS MAPA */

void mapa_mouseDown(Coordenada punto, InterfazMapa * interfaz) {
    Mapa * mapa = interfaz->getMapa();
    Coordenada puntoCorregido;
```

```
const Elemento * elemento;

if((mapa->entornoDePuntoDeEnlace(punto, &puntoCorregido) == true)) {
    if(mapa->getNroElementoParaCrear() > 2) { // si es longitudinal lo q se va a crear
        mapa->agregarElemento(puntoCorregido);
        if(elemento = mapa->getElementoSeleccionado() == NULL) return;
        if(elemento->es(LONGITUDINAL))
            mapa->setTipoSeleccion(ELEMENTO_SELECCIONADO_POR_EXTREM02);
        return;
    }
}

int resul = mapa->seleccionarElemento(punto);
elemento = mapa->getElementoSeleccionado();
if(elemento != NULL && elemento->estaBloqueado()) {
    mapa->setTipoSeleccion(ELEMENTO_NO_SELECCIONADO);
    return;
}
switch (resul) {

case ELEMENTO_NO_SELECCIONADO:
    mapa->agregarElemento(punto);
    elemento = mapa->getElementoSeleccionado();
    mapa->setMostrarPuntosDeEnlaceElemento(false);
    if(elemento == NULL) {
        interfaz->vectorDiferencia = punto - interfaz->getPosicion();
        return;
    }
    else if(elemento->es(LONGITUDINAL))
        mapa->setTipoSeleccion(ELEMENTO_SELECCIONADO_POR_EXTREM02);
    else if(elemento->es(CIRCULAR)) {
        mapa->setTipoSeleccion(ELEMENTO_SELECCIONADO_POR_CENTRO);
        interfaz->vectorDiferencia = punto - elemento->getPosicion();
    }
    break;

case ELEMENTO_SELECCIONADO_POR_CENTRO:
    if(mapa->elementoABorrar == true)
        mapa->eliminarElemento();
    else {
        mapa->setMostrarPuntosDeEnlaceElemento(false);
        interfaz->vectorDiferencia = punto - elemento->getPosicion();
    }
    break;
case ELEMENTO_SELECCIONADO_POR_EXTREM02:
    mapa->setMostrarPuntosDeEnlaceElemento(false);
    break;
}

void mapa_mouseOver(Coordenada click, InterfazMapa * interfaz) {
    Mapa * mapa = interfaz->getMapa();

    if(!interfaz->is_clicked() || !mapa->hayElementoSeleccionado()) return;

    Coordenada clickCorregido((long)0,(long)0);
    int tipoSeleccion = mapa->getTipoSeleccion();

    bool corregirClick = mapa->entornoDePuntoDeEnlace(click, &clickCorregido);

    Coordenada punto = (corregirClick == true)? clickCorregido : click;

    switch (tipoSeleccion) {

case ELEMENTO_SELECCIONADO_POR_CENTRO:
    mapa->moverElementoA(punto - interfaz->vectorDiferencia);
    break;
```

```
case ELEMENTO_SELECCIONADO_POR_EXTREMO2: // solo los longitudinales caen aqui
    if(mapa->getElementoSeleccionado()->es(COHETE))
        mapa->actualizarAngulo(punto);
    else
        mapa->actualizarModuloAngulo(punto);
    break;
}

void mapa_mouseUp(Coordenada click, InterfazMapa * interfaz) {
    Mapa * mapa = interfaz->getMapa();
    if(!mapa->hayElementoSeleccionado()) return;
    mapa->actualizarEnlaces();
    mapa->setMostrarPuntosDeEnlaceElemento(true);
}

/* EVENTOS BOTON MASA */

void botonMasa_mouseDown(Coordenada cord, Boton * boton) {

    Mapa * miMapa = boton->getMapa();
    miMapa->elementoABorrar = false;

    if(miMapa->getNroElementoParaCrear() != MASA) {
        miMapa->setNroElementoParaCrear(MASA);
    }else{
        miMapa->setNroElementoParaCrear(NINGUN_ELEMENTO);
    }
}

/* EVENTOS BOTON RUEDA */

void botonRueda_mouseDown(Coordenada cord, Boton * boton) {

    Mapa * miMapa = boton->getMapa();
    miMapa->elementoABorrar = false;

    if(miMapa->getNroElementoParaCrear() != RUEDA) {
        miMapa->setNroElementoParaCrear(RUEDA);
    }else{
        miMapa->setNroElementoParaCrear(NINGUN_ELEMENTO);
    }
}

/* EVENTOS BOTON PUNTOFIJO */

void botonPuntoFijo_mouseDown(Coordenada cord, Boton * boton) {

    Mapa * miMapa = boton->getMapa();
    miMapa->elementoABorrar = false;

    if(miMapa->getNroElementoParaCrear() != PUNTO_FIJO) {
        miMapa->setNroElementoParaCrear(PUNTO_FIJO);
    }else{
        miMapa->setNroElementoParaCrear(NINGUN_ELEMENTO);
    }
}

/* EVENTOS BOTON BARRA */

void botonBarra_mouseDown(Coordenada cord, Boton * boton) {

    Mapa * miMapa = boton->getMapa();
    miMapa->elementoABorrar = false;

    if(miMapa->getNroElementoParaCrear() != BARRA) {
        miMapa->setNroElementoParaCrear(BARRA);
    }
}
```

```
    }else{
        miMapa->setNroElementoParaCrear(NINGUN_ELEMENTO);
    }
}

/* EVENTOS BOTON PLATAFORMA */

void botonPlataforma_mouseDown(Coordenada cord, Boton * boton) {

    Mapa * miMapa = boton->getMapa();
    miMapa->elementoABorrar = false;

    if(miMapa->getNroElementoParaCrear() != PLATAFORMA) {
        miMapa->setNroElementoParaCrear(PLATAFORMA);
    }else{
        miMapa->setNroElementoParaCrear(NINGUN_ELEMENTO);
    }

}

/* EVENTOS BOTON SOGA */

void botonSoga_mouseDown(Coordenada cord, Boton * boton) {

    Mapa * miMapa = boton->getMapa();
    miMapa->elementoABorrar = false;

    if(miMapa->getNroElementoParaCrear() != SOGA) {
        miMapa->setNroElementoParaCrear(SOGA);
    }else{
        miMapa->setNroElementoParaCrear(NINGUN_ELEMENTO);
    }

}

/* EVENTOS BOTON CINTA */

void botonCinta_mouseDown(Coordenada cord, Boton * boton) {

    Mapa * miMapa = boton->getMapa();
    miMapa->elementoABorrar = false;

    if(miMapa->getNroElementoParaCrear() != CINTA) {
        miMapa->setNroElementoParaCrear(CINTA);
    }else{
        miMapa->setNroElementoParaCrear(NINGUN_ELEMENTO);
    }

}

/* EVENTOS BOTON ZONA_LLEGADA */

void botonZonaLlegada_mouseDown(Coordenada cord, Boton * boton) {

    Mapa * miMapa = boton->getMapa();
    miMapa->elementoABorrar = false;

    if(miMapa->getNroElementoParaCrear() != ZONA_LLEGADA) {
        miMapa->setNroElementoParaCrear(ZONA_LLEGADA);
    }else{
        miMapa->setNroElementoParaCrear(NINGUN_ELEMENTO);
    }

}

/* EVENTOS BOTON COHETE */

void botonCohete_mouseDown(Coordenada cord, Boton * boton) {

    Mapa * miMapa = boton->getMapa();
    miMapa->elementoABorrar = false;
```

```
if(miMapa->getNroElementoParaCrear() != COHETE) {
    miMapa->setNroElementoParaCrear(COHETE);
} else{
    miMapa->setNroElementoParaCrear(NINGUN_ELEMENTO);
}

/*
 * EVENTOS BOTON BORRAR_TODO */
void botonBorrarTodo_mouseDown(Coordenada cord, Boton * boton) {

    Mapa * miMapa = boton->getMapa();
    miMapa->elementoABorrar = false;
    miMapa->setNroElementoParaCrear(NINGUN_ELEMENTO);
    miMapa->eliminarTodosLosElementos();
}

/*
 * EVENTOS BOTON BORRAR_ELEMENTO */
void botonBorrarElemento_mouseDown(Coordenada cord, Boton * boton) {

    Mapa * miMapa = boton->getMapa();

    if(miMapa->elementoABorrar == false)
        miMapa->elementoABorrar = true;
    else
        miMapa->elementoABorrar = false;
    miMapa->setNroElementoParaCrear(NINGUN_ELEMENTO);
}

/*
 * EVENTOS BOTON PLAY */
void botonPlay_mouseDown(Coordenada cord, Boton * boton) {
    Mapa * miMapa = boton->getMapa();
    SMSApp app(boton->getSmsState(), miMapa);
    app.run();
}

/*
 * EVENTOS BOTON SALIR */
void botonSalir_mouseDown(Coordenada cord, Boton * boton) {
    boton->enviarSenialSalida();
}

/*
 * EVENTOS BOTON SMS */
void botonSms_mouseDown(Coordenada cord, Boton * boton) {
    boton->enviarSenialSms();
}

/*
 * EVENTOS BOTON GUARDAR */
void botonGuardar_mouseDown(Coordenada cord, Boton * boton) {
    Mapa * miMapa = boton->getMapa();
    if(miMapa->esCorrecto()) {
        boton->getMapa()->setGuardar(true);
        boton->enviarSenialSalida();
    }
}
```

```
#include "InterfazMapa.h"
#include <iostream>
#include <list>
#include "Elemento.h"
#include "Barra.h"
#include "Masa.h"
#include "Rueda.h"
#include <list>
#include "PuntoDeEnlace.h"

using namespace std;

InterfazMapa::InterfazMapa(const string & rutaImgDefecto,
    const string & rutaImgClickDown, Mapa * mapa) :
    PanelContenedor("Mapa", rutaImgDefecto, rutaImgDefecto, rutaImgDefecto) {
    this->mapa = mapa;
    pf_rutinaMouseDown = NULL;
    pf_rutinaMouseOver = NULL;
    pf_rutinaMouseUp = NULL;

    loadTexture("./ima/mapa/ball.bmp", TEXTURA_MASA);
    loadTexture("./ima/mapa/punto_fijo.bmp", TEXTURA_PUNTO_FIJO);
    loadTexture("./ima/mapa/lona.bmp", TEXTURA_CINTA);
    loadTexture("./ima/mapa/rope.bmp", TEXTURA_SOGA);
    loadTexture("./ima/mapa/metal.bmp", TEXTURA_BARRA);
    loadTexture("./ima/mapa/rocket.bmp", TEXTURA_COHETE);
    loadTexture("./ima/mapa/platform.bmp", TEXTURA_PLATAFORMA);
    loadTexture("./ima/mapa/rocket.bmp", TEXTURA_RUEDA);
    loadTexture("./ima/mapa/llegada.bmp", TEXTURA_ZONA_LLEGADA);

}

InterfazMapa::InterfazMapa(const string & rutaImgDefecto,
    const string & rutaImgClickDown, Mapa * mapa, double zoom) :
    PanelContenedor("Mapa", rutaImgDefecto, rutaImgDefecto, rutaImgDefecto, zoom) {
    this->mapa = mapa;
    pf_rutinaMouseDown = NULL;
    pf_rutinaMouseOver = NULL;
    pf_rutinaMouseUp = NULL;

    loadTexture("./ima/mapa/ball.bmp", TEXTURA_MASA);
    loadTexture("./ima/mapa/punto_fijo.bmp", TEXTURA_PUNTO_FIJO);
    loadTexture("./ima/mapa/lona.bmp", TEXTURA_CINTA);
    loadTexture("./ima/mapa/rope.bmp", TEXTURA_SOGA);
    loadTexture("./ima/mapa/metal.bmp", TEXTURA_BARRA);
    loadTexture("./ima/mapa/rocket.bmp", TEXTURA_COHETE);
    loadTexture("./ima/mapa/platform.bmp", TEXTURA_PLATAFORMA);
    loadTexture("./ima/mapa/rocket.bmp", TEXTURA_RUEDA);
    loadTexture("./ima/mapa/llegada.bmp", TEXTURA_ZONA_LLEGADA);

}

InterfazMapa::~InterfazMapa() {

}

void InterfazMapa::evento_MouseDown(Coordenada punto) {
    PanelContenedor::evento_MouseDown(punto);
    if (pf_rutinaMouseDown != NULL)
        pf_rutinaMouseDown(punto, this);
}

void InterfazMapa::evento_MouseUp(Coordenada punto) {
    PanelContenedor::evento_MouseUp(punto);
    if (pf_rutinaMouseUp != NULL)
        pf_rutinaMouseUp(punto, this);
}
```

```
void InterfazMapa::evento_MouseOver(Coordenada punto) {
    PanelContenedor::evento_MouseOver(punto);
    if (pf_rutinaMouseOver != NULL)
        pf_rutinaMouseOver(punto, this);
}

void InterfazMapa::setRutinaMouseDown(void (*rutina)(Coordenada, InterfazMapa *)) {
    pf_rutinaMouseDown = rutina;
}

void InterfazMapa::setRutinaMouseUp(void (*rutina)(Coordenada, InterfazMapa *)) {
    pf_rutinaMouseUp = rutina;
}

void InterfazMapa::setRutinaMouseOver(void (*rutina)(Coordenada, InterfazMapa *)) {
    pf_rutinaMouseOver = rutina;
}

Mapa * InterfazMapa::getMapa() const {
    return mapa;
}

void InterfazMapa::setMapa(Mapa* mapa)
{
    this->mapa = mapa;
}

void InterfazMapa::imprimir() {
    PanelContenedor::imprimir();

    list<Elemento*> lista = this->mapa->getListaElementos();
    list<Elemento*>:: iterator it;

    for (it = lista.begin(); it != lista.end(); it++) {
        if ((*it)->es(CIRCULAR))
            imprimirElementoCircular( (ElementoCircular*) (*it) );

        else if ((*it)->es(LONGITUDINAL))
            imprimirElementoLongitudinal( (ElementoLongitudinal*) (*it) );
        else if ((*it)->es(ZONA_LLEGADA))
            imprimirZonaLlegada( (Llegada*) (*it) );
    }
}

void InterfazMapa::imprimirElementoCircular(ElementoCircular * circular) {
    ulong radio = circular->getMagnitud();
    int textura= TEXTURA_MASA;

    if (circular->es(MASA))
        textura = TEXTURA_MASA;
    if (circular->es(PUNTO_FIJO)) {
        textura = TEXTURA_PUNTO_FIJO;
        radio = 4;
    }
    if (circular->es(PUNTO_ENLACE)) {
        textura = TEXTURA_PUNTO_FIJO;
        radio = 3;
    }
    if (circular->es(RUEDA))
        textura = TEXTURA_RUEDA;

    glBindTexture(GL_TEXTURE_2D, _texturas[textura]);
    glBegin(GL_TRIANGLE_FAN);

    double x, y;
    x = cambioX(circular->getPosicion().x) + getPosicion().x/TX;
    y = cambioY(circular->getPosicion().y) - getPosicion().y/TY;
```

```
for (int j = 0; j <= POLY_SIDES; j++) {  
    glTexCoord2i( 0, 0);  
    glVertex3f(x, y, 0.0f);  
    glTexCoord2i( 1, 0);  
    glVertex3f(x, y, 0.0f);  
  
    glTexCoord2i( 1, 1);  
    glVertex3f(x + radio/TX * cos(j * (2 * PI  
        / POLY_SIDES)), y + radio/TY* sin(j * (2  
        * PI / POLY_SIDES)), 0.0f);  
  
    j++;  
    glTexCoord2i( 0, 1);  
    glVertex3f(x + radio/TX * cos(j * (2 * PI  
        / POLY_SIDES)), y + radio/TY * sin(j * (2  
        * PI / POLY_SIDES)), 0.0f);  
    j--;  
}  
  
}  
  
glEnd();  
  
if (circular->es(RUEDA) && circular->getMostrarPuntosDeEnlace()) {  
    list <PuntoDeEnlace*> _enlaces = ((Rueda*)circular)->getListaEnlaces();  
    list <PuntoDeEnlace*>::iterator it;  
    for(it = _enlaces.begin(); it != _enlaces.end() ; it++) {  
        imprimirElementoCircular(*it);  
    }  
}  
}  
  
}  
  
void InterfazMapa::imprimirElementoLongitudinal(ElementoLongitudinal * elem) {  
    int punta = 0;  
    int textura= TEXTURA_MASA;  
    if (elem->es(BARRA))  
        textura = TEXTURA_BARRA;  
    else if (elem->es(CINTA))  
        textura = TEXTURA_CINTA;  
    else if (elem->es(SOGA))  
        textura = TEXTURA_SOGA;  
    else if (elem->es(COHETE)) {  
        textura = TEXTURA_COHETE;  
        punta = 10;  
    }  
    else if (elem->es(PLATAFORMA))  
        textura = TEXTURA_PLATAFORMA;  
  
    Coordenada extrOpPrima = elem->getExtremoOpuesto() -  
    Coordenada((long) (punta * cos(Angulo::aRadianes(elem->getAngulo()))), -(long)( punta * sin  
(Angulo::aRadianes(elem->getAngulo()))));  
  
    double aX = cambioX(elem->getPosicion().x) - (elem->getAncho()/(2.0*TX)) * cos  
(Angulo::aRadianes(90-elem->getAngulo() ));  
    double aY = cambioY(elem->getPosicion().y) + (elem->getAncho()/(2.0*TY)) * sin  
(Angulo::aRadianes(90-elem->getAngulo() ));  
  
    double bX = cambioX(extrOpPrima.x) -  
        (elem->getAncho()/(2.0*TX)) * cos(Angulo::aRadianes(90-elem->getAngulo() ));  
    double bY = cambioY(extrOpPrima.y) +  
        (elem->getAncho()/(2.0*TY)) * sin(Angulo::aRadianes(90-elem->getAngulo() ));  
  
    double cX = (elem->getAncho()/TX) *  
        sin(Angulo::aRadianes(elem->getAngulo())) + aX;  
    double cY = aY - (elem->getAncho())/TY) *
```

```
        cos(Angulo::aRadianes(elem->getAngulo()));

double dX = cambioX(extrOpPrima.x) +
    (elem->getAncho()/(2.0*TX)) * sin(Angulo::aRadianes(elem->getAngulo()) );

double dY = cambioY(extrOpPrima.y) -
    (elem->getAncho()/(2.0*TX)) * cos(Angulo::aRadianes(elem->getAngulo()) );

aX += getPosicion().x/TX;
bX += getPosicion().x/TX;
cX += getPosicion().x/TX;
dX += getPosicion().x/TX;

aY -= getPosicion().y/TY;
bY -= getPosicion().y/TY;
cY -= getPosicion().y/TY;
dY -= getPosicion().y/TY;

glBindTexture(GL_TEXTURE_2D, _texturas[textura]);
glBegin(GL_QUADS);

glTexCoord2i( 0, 0);
glVertex3f(cX, cY, 0.0f);
glTexCoord2i( 1, 0);
glVertex3f(aX, aY, 0.0f);
glTexCoord2i( 1, 1);
glVertex3f(bX, bY, 0.0f);
glTexCoord2i( 0, 1);
glVertex3f(dX, dY, 0.0f);

glEnd();

if (elem->es(COHETE)) {
    glBegin(GL_QUADS);

    glTexCoord2i( 0, 0);
    glVertex3f(dX, dY, 0.0f);
    glTexCoord2i( 1, 0);
    glVertex3f(bX, bY, 0.0f);
    glTexCoord2i( 1, 1);
    glVertex3f(cambioX(elem->getPosicion().x + elem->getMagnitud() * cos(-Angulo::aRadianes(elem->getAngulo()) + getPosicion().x), cambioY(elem->getPosicion().y + elem->getMagnitud() * sin(-Angulo::aRadianes(elem->getAngulo()) + getPosicion().y), 0.0f));
    glTexCoord2i( 0, 1);
    glVertex3f(cambioX(elem->getPosicion().x + elem->getMagnitud() * cos(-Angulo::aRadianes(elem->getAngulo()) + getPosicion().x), cambioY(elem->getPosicion().y + elem->getMagnitud() * sin(-Angulo::aRadianes(elem->getAngulo()) + getPosicion().y), 0.0f));

    glEnd();
}

if(elem->getMostrarPuntosDeEnlace() ) {
    list <PuntoDeEnlace*> _enlaces = elem->getListaEnlaces();
    list <PuntoDeEnlace*>::iterator it;
    for(it = _enlaces.begin(); it != _enlaces.end() ; it++) {
        imprimirElementoCircular(*it);
    }
}

void InterfazMapa::imprimirZonaLlegada(Llegada * llegada) {
    glColor4d(1.0, 1.0, 4.0, 0.4);
    glEnable (GL_BLEND);
    glBlendFunc (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
    glBindTexture( GL_TEXTURE_2D, _texturas[TEXTURA_ZONA_LLEGADA]);
    glBegin(GL_QUADS);

    glTexCoord2i( 0, 0);
    glVertex3f(cambioX(llegada->getPosicion().x + getPosicion().x), cambioY(llegada->getPosicion()
331
```

```
().y + getPosicion().y), 0.0f);
    glVertex3f(cambioX(llegada->getPosicion().x + getPosicion().x, cambioY(llegada->getPosicion()
() .y + getPosicion().y + (long)llegada->getMagnitud(), 0.0f);
    glVertex3f(cambioX(llegada->getPosicion().x + getPosicion().x + (long)llegada->getMagnitud()),
cambioY(llegada->getPosicion().y + getPosicion().y + (long)llegada->getMagnitud(), 0.0f);
    glVertex3f(cambioX(llegada->getPosicion().x + getPosicion().x + (long)llegada->getMagnitud()),
cambioY(llegada->getPosicion().y + getPosicion().y), 0.0f);

    glEnd();
    glColor4d(1.0, 1.0, 1.0, 1.0);
}
```

```
#include "Llegada.h"

Llegada::Llegada(Coordenada posicion) : Elemento ("ZonaLlegada", posicion, 80, 0) {

}

int Llegada::seleccionar(Coordenada punto) const {
    if((punto.x > getPosicion().x) && (punto.x < (getPosicion().x + (long)getMagnitud())))
        if((punto.y > getPosicion().y) && (punto.y < (getPosicion().y + (long)getMagnitud())))
            return ELEMENTO_SELECCIONADO_POR_CENTRO;
    return ELEMENTO_NO_SELECCIONADO;
}
```

```
#include "PanelContenedor.h"
#include "loader.h"
#include "Angulo.h"
#include "constantes.h"
#include "constantesnico.h"

#include <ctime>
#include <list>
#define ROPES 5
#define LONAS 6
#define RUEDAS 2
#define COHETES 7
#define METAL 3
#define PLATAFORMAS 4
#define PTO_FIJO 1
#define TX 300.0
#define TB 600.0
#define TY 300.0
using namespace std;

void asociateSpring(SpringMassSystem& sms, int spring, int elemento){
    ElementManager* manager;
    manager = sms.getElementManager();
    manager->addSpringElement(spring, elemento);
}

unsigned calculaComp(double largo,double &sep){
    double min = 0.03;
    unsigned i = 0;
    while (true){
        if ( (i*min<largo) && ((i+1)*min>=largo) )
            break;
        i++;
    }
    sep = largo / i;
    return i;
}

void drawCintaLona(SpringMassSystem& sms, double x,double y,double largo,double angulo){
    ElementManager* manager;
    manager = sms.getElementManager();

    int ini, fin,fspr;
    double sep = 0.04;
    int longitud = calculaComp(largo,sep);

    fspr = sms.getNumSprings();
    ini = sms.getNumMasses();
    fin = ini + longitud;
    for (int i=0;i<longitud;i++)
        sms.addCollMass(2.0, Vector2<>(x+i*sep*cos(angulo), y+i*sep*sin(angulo)), 0.01);

    for (int i=ini ;i<fin ;i++)
        sms.addSpring(i,i+1,LONA_K,0);

    manager->addLona(ini,fin);
    sms.addEnganche(ini);
    sms.addEnganche(fin);

    ini = (manager->getLonas()).size()-1;
    for (size_t i=fspr; i<sms.getNumSprings();i++)
        asociateSpring(sms,i,ini);
}

void drawMetalBar(SpringMassSystem& sms, double x,double y,double largo,double angulo){
    ElementManager* manager;
    manager = sms.getElementManager();
    double sep = 0.04;
    int longitud = calculaComp(largo,sep);

    int ini,fin,first,last;
    ini = sms.getNumMasses();
```

```
first = ini;
fin = ini + longitud;

for (int i=0;i<=longitud;i++)
    sms.addMass(2.0, Vector2<>(x+i*sep*cos(angulo), y+i*sep*sin(angulo)));

last = sms.getNumMasses()-1;
for (int i=ini ;i<fin ;i++)
    sms.addSpring(i,i+1,METAL_K,0);

x += METAL_WIDTH * sin(angulo);
y -= METAL_WIDTH * cos(angulo) ;

ini = sms.getNumMasses();
fin = ini + longitud;

for (int i=0;i<=longitud;i++)
    sms.addMass(2.0, Vector2<>(x+i*sep*cos(angulo), y+i*sep*sin(angulo)));

for (int i=ini ;i<fin ;i++)
    sms.addSpring(i,i+1,METAL_K,0);

for (int i=ini ;i<=fin ;i++)
    sms.addSpring(i,i-longitud-1,METAL_K,0);
for (int i=ini+1 ;i<=fin ;i++)
    sms.addSpring(i,i-longitud-2,METAL_K,0);

manager->addMetalBar(first,sms.getNumMasses()-1);
sms.addEnganche(first);
sms.addEnganche(last);
}

void drawPlataforma(SpringMassSystem& sms, double x,double y,double largo,double angulo){
    ElementManager* manager;
    manager = sms.getElementManager();
    double sep = 0.04;
    int longitud = calculaComp(largo,sep);

    int ini,fin,first,last;
    ini = sms.getNumMasses();
    first = ini;
    fin = ini + longitud;

    for (int i=0;i<=longitud;i++)
        sms.addCollMass(2.0, Vector2<>(x+i*sep*cos(angulo), y+i*sep*sin(angulo)), 0.01);

    last = sms.getNumMasses()-1;
    for (int i=ini ;i<fin ;i++)
        sms.addSpring(i,i+1,METAL_K,0);

    x += METAL_WIDTH * sin(angulo);
    y -= METAL_WIDTH * cos(angulo) ;

    ini = sms.getNumMasses();
    fin = ini + longitud;

    for (int i=0;i<=longitud;i++)
        sms.addMass(2.0, Vector2<>(x+i*sep*cos(angulo), y+i*sep*sin(angulo)));

    for (int i=ini ;i<fin ;i++)
        sms.addSpring(i,i+1,METAL_K,0);

    for (int i=ini ;i<=fin ;i++)
        sms.addSpring(i,i-longitud-1,METAL_K,0);
    for (int i=ini+1 ;i<=fin ;i++)
        sms.addSpring(i,i-longitud-2,METAL_K,0);

    manager->addPlataforma(first,sms.getNumMasses()-1);
    sms.addEnganche(first);
    sms.addEnganche(last);
```

```
}

void drawRueda(SpringMassSystem& sms, double x,double y,double radio){
    ElementManager* manager;
    manager = sms.getElementManager();

    unsigned sides = 12 ;
    int ini = sms.getNumMasses();
    sms.addCollMass( 0,Vector2<>(x,y),0.05);
    double dx,dy;
    for (unsigned j = 0; j < sides; j++){
        dx = radio * cos(j * (2.0 * PI / sides));
        dy = radio * sin(j * (2.0 * PI / sides));
        sms.addCollMass( 1.5,
            Vector2<>(x + dx, y + dy),
            0.01);
    }
    for (unsigned j = ini+1; j < ini + sides; j++){
        sms.addSpring(j,j+1,RUEDA_K,0);
        sms.addSpring(ini,j,RUEDA_K,0);
    }

    sms.addSpring(ini+1,ini + sides,RUEDA_K,0);
    sms.addSpring(ini,ini + sides,RUEDA_K,0);
    manager->addRueda(ini,sms.getNumMasses()-1);

    for (size_t i = ini + 1; i<=(ini + sides); i++)
        sms.addEnganche(i);
}

void drawCohete(SpringMassSystem& sms, double x,double y,double largo,double angulo){
    ElementManager* manager;
    double xprime,yprime;
    x-=ROCKET_WIDTH*0.5*sin(angulo);
    y+=ROCKET_WIDTH*0.5*cos(angulo);

    xprime = x;
    yprime = y;

    manager = sms.getElementManager();
    int ini, fin,first;
    ini = sms.getNumMasses();
    first = ini;
    double sep = 0.04;
    int longitud = calculaComp(largo,sep);

    fin = ini + longitud - 1;
    for (int i=0;i<longitud;i++)
        sms.addCollMass(1, Vector2<>(x+i*sep*cos(angulo), y+i*sep*sin(angulo)), 0.01);

    for (int i=ini ;i<fin ;i++)
        sms.addSpring(i,i+1,ROCKET_K,0);

    ini = sms.getNumMasses();
    fin = ini + longitud - 1;

    xprime = x + ROCKET_WIDTH * sin(angulo);
    yprime = y - ROCKET_WIDTH * cos(angulo);

    for (int i=0;i<longitud;i++)
        sms.addCollMass(1, Vector2<>(xprime+i*sep*cos(angulo), yprime+i*sep*sin(angulo)),
0.01);

    for (int i=ini ;i<fin ;i++)
        sms.addSpring(i,i+1,ROCKET_K,0);

    for (int i=ini ;i<=fin ;i++)
        sms.addSpring(i,i-longitud,ROCKET_K,0);

    for (int i=ini ;i<fin ;i++)
```

```
        sms.addSpring(i,i-longitud+1,ROCKET_K,0);

xprime = x+ROCKET_WIDTH*0.5*sin(angulo);
yprime = y-ROCKET_WIDTH*0.5*cos(angulo);

sms.addCollMass(1, Vector2<>(xprime+longitud*sep*cos(angulo), yprime+longitud*sep*sin(angulo)), 0.03);
ini = sms.getNumMasses();
sms.addSpring(ini-1,first+longitud-1,ROCKET_K,0);
sms.addSpring(ini-1,first+2*longitud-1,ROCKET_K,0);

sms.addCollMass(1, Vector2<>(xprime, yprime), 0.01);
sms.addSpring(ini,first,ROCKET_K,0);
sms.addSpring(ini,first+longitud,ROCKET_K,0);
manager->addCohete(first,sms.getNumMasses()-1);
sms.addEnganche(sms.getNumMasses()-1);
sms.addEnganche(sms.getNumMasses()-2);

}

void drawRope(SpringMassSystem& sms, double x,double y,double largo,double angulo){
    ElementManager* manager;
    manager = sms.getElementManager();
    int ini, fin;
    double sep = 0.04;
    int longitud = calculaComp(largo,sep);

    ini = sms.getNumMasses();
    fin = ini + longitud;

    for (int i=0;i<=longitud;i++)
        sms.addMass(3, Vector2<>(x+i*sep*cos(angulo), y+i*sep*sin(angulo)));

    for (int i=ini ;i<fin ;i++)
        sms.addSpring(i,i+1,ROPE_K,0);

    manager->addRope(ini,fin);
    sms.addEnganche(ini);
    sms.addEnganche(fin);
}

void drawFixedSpot(SpringMassSystem& sms, double x,double y){
    ElementManager* manager;
    manager = sms.getElementManager();
    sms.addMass(0,Vector2<>(x, y));
    manager->addPuntoFijo(sms.getNumMasses()-1,sms.getNumMasses()-1);
    sms.addEnganche(sms.getNumMasses()-1);
}

void drawCrashableFixedSpot(SpringMassSystem& sms, double x,double y){
    sms.addCollMass(0,Vector2<>(x, y),0.03);
}

void drawBall(SpringMassSystem& sms, double x,double y,double radio){
    ElementManager* manager;
    manager = sms.getElementManager();
    unsigned sides = 16 ;
    int ini = sms.getNumMasses();
    sms.addCollMass( 0.5,Vector2<>(x,y),0.05);
    double dx,dy;
    for (unsigned j = 0; j < sides; j++){
        dx = radio * cos(j * (2.0 * PI / sides));
        dy = radio * sin(j * (2.0 * PI / sides));
        sms.addCollMass( 1.5,
                        Vector2<>(x + dx, y + dy),
                        0.009);
    }
    for (unsigned j = ini+1; j < ini + sides; j++){
        sms.addSpring(j,j+1,BALL_K,0);
        sms.addSpring(ini,j,BALL_K,0);
    }
}
```

```
sms.addSpring(ini+1,ini + sides,BALL_K,0);
sms.addSpring(ini,ini + sides,BALL_K,0);
manager->addMainBall(ini,sms.getNumMasses()-1);
}

void drawZonaLlegada(SpringMassSystem& sms,double x ,double y,double width,double heighth ){
    ElementManager* manager;
    manager = sms.getElementManager();
    manager->addZonaLlegada(x,y,width,heighth);
}

void fixMass(SpringMassSystem& sms,int pos){
    sms.fixMass(pos);
}

double cambioX(long x) {
    return (x - 1000/2.0)/TX;
}

double cambioY(long y) {
    return (-y + 600/2.0)/TY;
}

void instanciar(SpringMassSystem& sms)
{
    Mapa* mapa = sms.getMapa();

    list <Elemento*> elementos = mapa->getListaElementos();
    list <Elemento*>::iterator it;

    for(it = elementos.begin(); it != elementos.end(); it++) {
        if((*it)->es(MASA))
            drawBall(sms, cambioX((*it)->getPosicion().x) ,cambioY((*it)->getPosicion().y),
(*it)->getMagnitud()/TX);
        if((*it)->es(RUEDA))
            drawRueda(sms, cambioX((*it)->getPosicion().x) ,cambioY((*it)->getPosicion()
().y),(*it)->getMagnitud()/TX);
        if((*it)->es(SOGA))
            drawRope(sms, cambioX((*it)->getPosicion().x) ,cambioY((*it)->getPosicion(),
(*it)->getMagnitud()/TX,Angulo::aRadianes((*it)->getAngulo())));
        if((*it)->es(CINTA))
            drawCintaLona(sms, cambioX((*it)->getPosicion().x) ,cambioY((*it)->getPosicion()
().y),(*it)->getMagnitud()/TX,Angulo::aRadianes((*it)->getAngulo())));
        if((*it)->es(PLATAFORMA))
            drawPlataforma(sms, cambioX((*it)->getPosicion().x) ,cambioY((*it)->getPosicion()
().y),(*it)->getMagnitud()/TX,Angulo::aRadianes((*it)->getAngulo())));
        if((*it)->es(BARRA))
            drawMetalBar(sms, cambioX((*it)->getPosicion().x) ,cambioY((*it)->getPosicion()
().y),(*it)->getMagnitud()/TX,Angulo::aRadianes((*it)->getAngulo())));
        if((*it)->es(COHETE))
            drawCohete(sms, cambioX((*it)->getPosicion().x) ,cambioY((*it)->getPosicion()
().y),(*it)->getMagnitud()/TX,Angulo::aRadianes((*it)->getAngulo())));
        if((*it)->es(PUNTO_FIJO))
            drawFixedSpot(sms, cambioX((*it)->getPosicion().x) ,cambioY((*it)->getPosicion()
().y));
        if((*it)->es(ZONA_LLEGADA))
            drawZonaLlegada(sms,cambioX((*it)->getPosicion().x) ,cambioY((*it)->getPosicion()
().y),(*it)->getMagnitud()/TX,(*it)->getMagnitud()/TX );
    }
    sms.checkRoutine();
}

void (*instance[])(SpringMassSystem&) = {instanciar};

void load(SpringMassSystem& sms)
{
```

```
    srand(static_cast<unsigned>(time(0)));

    int n = sizeof(instance) / sizeof(instance[0]);

    instance[rand() % n](sms);
}
```

```
#include "InterfazEdicion.h"
#include "Mapa.h"
#include <glib.h>
#include <glib/gprintf.h>
#include <gtk/gtk.h>
#include <string>
#include "ParserXML.h"
#include "VentanaConfiguracion.h"
#include <sys/stat.h>
using namespace std;

VentanaConfiguracion* configuracion;

// estructura de datos auxiliar para pasaje de parámetros
typedef struct datosAbrir
{
    GtkWidget* ventanaPpal;
    GtkWidget* entrada;
    GtkWidget* ventanaAbrir;
}datosAbrir;

/* Creación de escenario desde cero */
void crear_nuevo(GtkWidget *widget, gpointer data)
{
    GtkWidget* ventana = (GtkWidget*) data;
    Mapa* mapa = new Mapa(); // mapa nuevo

    // se ejecuta la edición
    InterfazEdicion inter(mapa);
    inter.comenzar();

    // si el usuario pidió guardar se pasa a la configuración
    if (mapa->debeGuardarse())
        configuracion = new VentanaConfiguracion(mapa,ventana);
    else
    {
        delete mapa;
        gtk_widget_destroy(ventana);
    }
}

/* Devuelve true si el texto es vacío o solamente tiene espacios */
bool textoVacio(const string& texto)
{
    unsigned int i;
    bool retorno = true;
    for (i = 0 ; i < texto.length() ; i++)
    {
        if (texto[i] != ' ')
        {
            retorno = false;
            break;
        }
    }
    return retorno;
}

/* Analiza si existe el archivo del escenario dado
 * Devuelve true si existe, false si no */
bool existeEscenario(const string& escenario)
{
    struct stat stFileInfo;
    string strFilename;
    int intStat;

    strFilename = "escenarios/" + escenario + ".xml";

    intStat = stat(strFilename.c_str(),&stFileInfo);
    if(intStat != 0)
        return false;

    return true;
```

```
}

/* Se intenta abrir un escenario existente */
void abrir_escenario(GtkWidget *widget, gpointer data)
{
    datosAbrir* datos = (datosAbrir*) data;

    string nomArchivo;
    string escenario = gtk_entry_get_text(GTK_ENTRY(datos->entrada));

    gtk_entry_set_text(GTK_ENTRY(datos->entrada), "");

    if (textoVacio(escenario))
        return;

    //validar archivo inexistente
    if (!existeEscenario(escenario))
    {
        GtkWidget *alerta = gtk_dialog_new ();
        GtkWidget* label = gtk_label_new ("Escenario inexistente");

        gtk_box_pack_start (GTK_BOX (GTK_DIALOG (alerta)->vbox), label, FALSE, FALSE, 0);

        GtkWidget* boton = gtk_button_new_from_stock("Aceptar");
        gtk_box_pack_start (GTK_BOX (GTK_DIALOG (alerta)->action_area), boton, FALSE, FALSE, 0);

        gtk_window_set_title(GTK_WINDOW(alerta), "TatuCarreta");

        gtk_window_set_transient_for (GTK_WINDOW(alerta),GTK_WINDOW(datos->ventanaAbrir));
        gtk_window_set_position(GTK_WINDOW(alerta),GTK_WIN_POS_CENTER_ON_PARENT);
        gtk_window_set_title(GTK_WINDOW(alerta), "Error");

        gtk_signal_connect_object (GTK_OBJECT (boton), "clicked", GTK_SIGNAL_FUNC
(gtk_widget_destroy), GTK_OBJECT
(alerta));

        gtk_widget_show_all (alerta);
        return;
    }

    nomArchivo = "escenarios/" + escenario + ".xml";
    ParserXML par;
    Mapa* mapa;

    // se obtiene el mapa desde el parser
    mapa = par.obtenerMapaArchivo(nomArchivo);

    gtk_widget_destroy(datos->ventanaAbrir);

    // inicia edición
    InterfazEdicion inter(mapa);
    inter.comenzar();

    // si el usuario pide guardarse, se llama a la ventana de configuración
    if (mapa->debeGuardarse())
        configuracion = new VentanaConfiguracion(mapa,datos->ventanaPpal);
    else
    {
        delete mapa;
        gtk_widget_destroy(datos->ventanaPpal);
    }

    delete datos;
}

/* Ventana que permite abrir un escenario existente */
void abrir_existente(GtkWidget *widget, gpointer data)
{
    configuracion = NULL;

    GtkWidget* ventana = (GtkWidget*) data;
```

```
GtkWidget* ventanaAbrir = gtk_window_new(GTK_WINDOW_TOPLEVEL);
GtkWidget* contenedor = gtk_vbox_new(FALSE, 10);
GtkWidget* label = gtk_label_new("Nombre de escenario");
GtkWidget* botonAceptar = gtk_button_new_from_stock("Abrir");

GtkWidget* entrada = gtk_entry_new();
gtk_entry_set_width_chars(GTK_ENTRY(entrada), 30);

gtk_window_set_transient_for (GTK_WINDOW(ventanaAbrir), GTK_WINDOW(ventana));
gtk_window_set_position(GTK_WINDOW(ventanaAbrir), GTK_WIN_POS_CENTER_ON_PARENT);
gtk_window_set_title(GTK_WINDOW(ventanaAbrir), "TatuCarreta");

gtk_container_add(GTK_CONTAINER(ventanaAbrir), contenedor);

gtk_box_pack_start (GTK_BOX(contenedor), label, FALSE, FALSE, 10);
gtk_box_pack_start (GTK_BOX(contenedor), entrada, TRUE, TRUE, 10);
gtk_box_pack_start (GTK_BOX(contenedor), botonAceptar, FALSE, FALSE, 10);

datosAbrir* datos = new datosAbrir();
datos->ventanaPpal = ventana;
datos->ventanaAbrir = ventanaAbrir;
datos->entrada = entrada;

g_signal_connect(G_OBJECT(botonAceptar), "clicked", G_CALLBACK(abrir_escenario), datos);

gtk_widget_show_all(ventanaAbrir);
}

/* Cierra el Gtk */
void destruir(GtkWidget *widget, gpointer data)
{
    gtk_main_quit();
}

int main(int argc, char* argv[])
{
    gtk_init(&argc, &argv);

    /* Ventana con menú principal */
    GtkWidget* ventana = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    GtkWidget* contenedor = gtk_vbox_new(FALSE, 10);
    GtkWidget* botonNuevo = gtk_button_new_from_stock("Crear nuevo escenario");
    GtkWidget* botonAbrir = gtk_button_new_from_stock("Abrir escenario existente");
    GtkWidget* botonCerrar = gtk_button_new_from_stock("Cerrar editor");

    gtk_window_set_position(GTK_WINDOW(ventana), GTK_WIN_POS_CENTER);
    gtk_container_set_border_width(GTK_CONTAINER(ventana), 10);
    gtk_window_set_title (GTK_WINDOW (ventana), "Editor :: TatuCarreta Run");
    gtk_window_set_default_size (GTK_WINDOW (ventana), 300, 100);
    gtk_window_set_resizable(GTK_WINDOW (ventana), FALSE);

    gtk_container_add(GTK_CONTAINER(ventana), contenedor);

    gtk_box_pack_start (GTK_BOX(contenedor), botonNuevo, FALSE, FALSE, 10);
    gtk_box_pack_start (GTK_BOX(contenedor), botonAbrir, FALSE, FALSE, 10);
    gtk_box_pack_start (GTK_BOX(contenedor), botonCerrar, FALSE, FALSE, 10);

    g_signal_connect(G_OBJECT(botonNuevo), "clicked", G_CALLBACK(crear_nuevo), ventana);
    g_signal_connect(G_OBJECT(botonAbrir), "clicked", G_CALLBACK(abrir_existente), ventana);
    g_signal_connect(G_OBJECT(botonCerrar), "clicked", G_CALLBACK(destruir), NULL);
    g_signal_connect(G_OBJECT(ventana), "destroy", G_CALLBACK(destruir), NULL);

    gtk_widget_show_all (ventana);

    gtk_main();

    if (configuracion != NULL)
        delete configuracion;
}
```

```
#include "Mapa.h"
#include "ElementoEnlazable.h"
#include "ElementoLongitudinal.h"
#include "PuntoDeEnlace.h"
#include <math.h>
#include "Angulo.h"
using namespace std;

Mapa::Mapa() {
    this->ancho = 1024;
    this->alto = 550;
    elemento = NULL;
    nroElementoParaCrear = NINGUN_ELEMENTO;
    tipoSeleccion = ELEMENTO_NO_SELECCIONADO;
    elementoABorrar = false;

    for(int i = 0; i < CANTIDAD_ELEMENTOS; i++) {
        elementosHabilitados[i] = true; // todos habilitados por defecto
        precio[i] = 0;
    }

    plata = 20000;
    puntos = -1;
    aGuardar = false;
    tiempoCohete = 0;
}

Mapa::Mapa(ulong ancho, ulong alto) {
    this->ancho = ancho;
    this->alto = alto;
    elemento = NULL;
    nroElementoParaCrear = NINGUN_ELEMENTO;
    tipoSeleccion = ELEMENTO_NO_SELECCIONADO;
    elementoABorrar = false;

    for(int i = 0; i < CANTIDAD_ELEMENTOS; i++) {
        elementosHabilitados[i] = true; // todos habilitados por defecto
        precio[i] = 10; // son gratis por default
    }

    aGuardar = false;
    plata = 20000;
    puntos = -1;
    tiempoCohete = 0;
}

Mapa::~Mapa() {
    eliminarTodosLosElementos();
}

ulong Mapa::getAncho() const {
    return ancho;
}

ulong Mapa::getAlto() const {
    return alto;
}

void Mapa::setNroElementoParaCrear(int elemento) {
    nroElementoParaCrear = elemento;
    if(elemento== NINGUN_ELEMENTO) elemento = NULL;
}

int Mapa::getNroElementoParaCrear() const {
    return nroElementoParaCrear;
}

list <Elemento*> Mapa::getListaElementos() const {
    return _elementos;
}
```

```
bool Mapa::agregarElemento(Coordenada posicion) {
    FabricaDeElementos fabrica;
    Elemento * aux;
    aux = fabrica.crear(nroElementoParaCrear, posicion);
    if (aux !=NULL) {
        _elementos.push_back(aux);
        elemento = aux;
        return true;
    } else
        return false;
}

size_t Mapa::getCantidadElementos() {
    return _elementos.size();
}

int Mapa::seleccionarElemento(Coordenada punto) {
    list<Elemento*>::iterator it;

    int resultado= ELEMENTO_NO_SELECCIONADO;

    for (it = _elementos.begin(); it != _elementos.end(); it++)
        if ((resultado = (*it)->seleccionar(punto)) != ELEMENTO_NO_SELECCIONADO) {
            elemento = (*it);
            break;
        }

    if(resultado == ELEMENTO_NO_SELECCIONADO) {
        elemento = NULL;
    }

    tipoSeleccion = resultado;
    return resultado;
}

bool Mapa::hayElementoSeleccionado() const {
    return ((elemento != NULL)? true : false);
}

int Mapa::getTipoSeleccion() const {
    return tipoSeleccion;
}

void Mapa::setTipoSeleccion(const int tipo) {
    tipoSeleccion = tipo;
}

const Elemento * Mapa::getElementoSeleccionado() const {
    return elemento;
}

bool Mapa::entornoDePuntoDeEnlace(Coordenada punto, Coordenada * puntoExacto) {

    list <ElementoEnlazable*> _enlaces = getTodosLosEnlaces();
    list <ElementoEnlazable*>::iterator it;

    for (it = _enlaces.begin(); it != _enlaces.end(); it++) {
        if((*it)->seleccionar(punto) != ELEMENTO_NO_SELECCIONADO) {
            *puntoExacto = (*it)->getPosicion();
            return true;
        }
    }

    return false;
}

void Mapa::setExtremo0puesto(const Coordenada punto) {
    if(elemento != NULL) {
        if(elemento->es(LONGITUDINAL))
            ((ElementoLongitudinal*)elemento)->setExtremo0puesto(punto);
    }
}
```

```
}

Coordenada Mapa::getExtremoOpuesto() const {
    if(elemento != NULL) {
        return ((ElementoLongitudinal*)elemento)->getExtremoOpuesto();
    }
    return Coordenada((long) -1,(long)-1);
}

bool Mapa::moverElementoA(const Coordenada posicion) {
    if(elemento != NULL)
        elemento->setPosicion(posicion);
    return true;
}

bool Mapa::rotarElementoA(long angulo) {
    Angulo::corregir(&angulo);
    if(elemento != NULL)
        elemento->setAngulo(angulo);
    return true;
}

bool Mapa::agrandarElementoA(long longitud) {
    if(elemento != NULL)
        elemento->setMagnitud(longitud);
    return true;
}

Coordenada Mapa::actualizarModuloAngulo(Cordenada click) {
    if(elemento != NULL) {
        if( (click - elemento->getPosicion()).modulo < LONGITUD_MINIMA)
            actualizarAngulo(click);
        else {
            if(elemento->es(BARRA) || elemento->es(PLATAFORMA)) {
                if( (click - elemento->getPosicion()).modulo < LONGITUD_MAXIMA)
                    ((ElementoLongitudinal *)elemento)->setExtremoOpuesto(click);
                else
                    actualizarAngulo2(click);
            } else
                ((ElementoLongitudinal *)elemento)->setExtremoOpuesto(click);
        }
    }
    return click;
}

Coordenada Mapa::actualizarAngulo(Cordenada click) {
    if(elemento != NULL) {
        int longitudMinima = LONGITUD_MINIMA;

        if(elemento->es(COHETE))
            longitudMinima = LONGITUD_COHETE;
        Coordenada resta = click - elemento->getPosicion();
        if(resta.angulo > 180)
            resta.setAngulo(-resta.angulo);
        resta.setModulo(longitudMinima);
        ((ElementoLongitudinal *)elemento)->setExtremoOpuesto(elemento->getPosicion() + resta);
    }
    return click;
}

Coordenada Mapa::actualizarAngulo2(Cordenada click) {
    if(elemento != NULL) {

        int longitudMinima = LONGITUD_MAXIMA;

        if(elemento->es(COHETE))
            longitudMinima = LONGITUD_COHETE;
        Coordenada resta = click - elemento->getPosicion();
        if(resta.angulo > 180)
```

```
        resta.setAngulo(-resta.angulo);
        resta.setModulo(longitudMinima);
        ((ElementoLongitudinal *)elemento)->setExtremoOpuesto(elemento->getPosicion() + resta);
    }
    return click;
}
void Mapa::eliminarElemento() {
    if(elemento != NULL) {
        _elementos.remove(elemento);
        delete elemento;
        elemento = NULL;
    }
}

void Mapa::eliminarTodosLosElementos() {
    list<Elemento*>::iterator it;
    for (it = _elementos.begin(); it != _elementos.end(); it++) {
        delete (*it);
    }
    nroElementoParaCrear = NINGUN_ELEMENTO;
    elemento = NULL;
    _elementos.clear();
}

void Mapa::actualizarEnlaces() {
    list<Elemento*>::iterator it;
    for (it = _elementos.begin(); it != _elementos.end(); it++) {
        if((*it)->es(LONGITUDINAL)) {
            ElementoLongitudinal * ele = (ElementoLongitudinal *) *it;
            ele->actualizarPuntosDeEnlace(1);
        }
        else if( (*it)->es(RUEDA)) {
            ((Rueda*)(*it))->actualizarPuntosDeEnlace();
        }
    }
}

list <ElementoEnlazable*> Mapa::getTodosLosEnlaces() {
    list <ElementoEnlazable*> lista;

    list<Elemento*>::iterator it;
    for (it = _elementos.begin(); it != _elementos.end(); it++) {
        if( (*it)->es(ENLAZABLE) )
            lista.push_back((ElementoEnlazable*) *it);
        else if( (*it)->es(LONGITUDINAL) ) {
            ElementoLongitudinal * eleLong = (ElementoLongitudinal *) *it;
            list <PuntoDeEnlace*> _puntosDeEnlace = eleLong->getListaEnlaces();
            list <PuntoDeEnlace*>::iterator it2;
            for(it2 = _puntosDeEnlace.begin(); it2 != _puntosDeEnlace.end(); it2++) {
                lista.push_back((ElementoEnlazable*) *it2);
            }
        }
        else if( (*it)->es(RUEDA) ) {
            list <PuntoDeEnlace*> _puntosDeEnlace = ((Rueda*)(*it))->getListaEnlaces();
            list <PuntoDeEnlace*>::iterator it2;
            for(it2 = _puntosDeEnlace.begin(); it2 != _puntosDeEnlace.end(); it2++) {
                lista.push_back((ElementoEnlazable*) *it2);
            }
        }
    }
    return lista;
}

void Mapa::imprimirTodosLosEnlaces() {
    list <ElementoEnlazable*> lista = getTodosLosEnlaces();

    list<ElementoEnlazable*>::iterator it;
    for (it = lista.begin(); it != lista.end(); it++) {
        (*it)->getPosicion().imprimir();
    }
}
```

```
        }
    }

void Mapa::imprimir() {
    cout << "PLATA: " << plata << endl;
    list<Elemento*>::iterator it;
    for (it = _elementos.begin(); it != _elementos.end(); it++) {
        (*it)->imprimir();
    }
}

bool Mapa::esCorrecto() {
    bool masa = false;
    bool llegada = false;

    list<Elemento*>::iterator it;
    for (it = _elementos.begin(); it != _elementos.end(); it++) {
        if( (*it)->es(MASA) ) masa = true;
        if( (*it)->es(ZONA_LLEGADA) ) llegada = true;
    }

    if(masa == true && llegada == true)
        return true;
    else
        return false;
}

void Mapa::setMostrarPuntosDeEnlaceElemento(const bool b) {
    if(elemento != NULL) {
        elemento->setMostrarPuntosDeEnlace(b);
    }
}

void Mapa::bloquearElementos() {
    list<Elemento*>::iterator it;
    for (it = _elementos.begin(); it != _elementos.end(); it++) {
        (*it)->bloquear();
    }
}

void Mapa::deshabilitarElemento(const int nroElemento) {
    elementosHabilitados[nroElemento] = false;
}

void Mapa::habilitarElemento(const int nroElemento) {
    elementosHabilitados[nroElemento] = true;
}

bool Mapa::elementoHabilitado(const int nroElemento) const {
    return elementosHabilitados[nroElemento];
}

void Mapa::setNombre(const string & nombre) {
    this->nombre = nombre;
}

string Mapa::getNombre() const {
    return nombre;
}

void Mapa::setPlata(const int plata) {
    this->plata = plata;
}

int Mapa::getPlata() const {
    return plata;
}

void Mapa::setPuntos(const int puntos) {
```

```
    this->puntos = puntos;
}

int Mapa::getPuntos() const {
    return puntos;
}

void Mapa::setGuardar(bool valor)
{
    aGuardar = valor;
}

bool Mapa::debeGuardarse()
{
    return aGuardar;
}

void Mapa::setPrecio(int nroElemento, int precio) {
    this->precio[nroElemento] = precio;
}

int Mapa::getPrecio(int nroElemento) const{
    return this->precio[nroElemento];
}

int Mapa::getTiempoCohete() const {
    return tiempoCohete;
}

void Mapa::setTiempoCohete(int tiempo) {
    this->tiempoCohete = tiempo;
}
```

```
#include "Masa.h"
using namespace std;

Masa::Masa() :
    ElementoCircular("Circular:Masa", Coordenada(long(100), long(100)), 10, 0) {}

Masa::Masa(Cordenada posicion, ulong magnitud) :
    ElementoCircular("Circular:Masa", posicion, magnitud, 0) {}

Masa::~Masa() {
```

```
#include "PanelContenedor.h"
#include <iostream>
#include "constantes.h"
using namespace std;
PanelContenedor::PanelContenedor(const string & nombre,
                                  const string & rutaImgDefecto, const string & rutaImgClickDown,
                                  const string & rutaImgMouseOver) {

    this->nombre = nombre;

    ancho = getAncho(rutaImgDefecto);
    alto = getAlto(rutaImgDefecto);

    loadTexture(rutaImgDefecto.c_str(), TEXTURA_DEFECTO);
    loadTexture(rutaImgClickDown.c_str(), TEXTURA_MOUSEDOWN);
    loadTexture(rutaImgMouseOver.c_str(), TEXTURA_MOUSEOVER);
    loadTexture(rutaImgMouseOver.c_str(), TEXTURA_INHABILITADO);
    texturaActual = TEXTURA_DEFECTO;

    this->id = 0;

    posicion.setX(0);
    posicion.setY(0);
    pantalla = NULL;
    visible = true;
    habilitado = true;
    clicked = false;
    mouseOver = false;
    TX = 300.0;
    TY = 300.0;
}

PanelContenedor::PanelContenedor(const string & nombre,
                                  const string & rutaImgDefecto, const string & rutaImgClickDown,
                                  const string & rutaImgMouseOver, double zoom) {

    this->nombre = nombre;

    ancho = getAncho(rutaImgDefecto);
    alto = getAlto(rutaImgDefecto);

    loadTexture(rutaImgDefecto.c_str(), TEXTURA_DEFECTO);
    loadTexture(rutaImgClickDown.c_str(), TEXTURA_MOUSEDOWN);
    loadTexture(rutaImgMouseOver.c_str(), TEXTURA_MOUSEOVER);
    loadTexture(rutaImgMouseOver.c_str(), TEXTURA_INHABILITADO);
    texturaActual = TEXTURA_DEFECTO;

    this->id = 0;

    posicion.setX(0);
    posicion.setY(0);
    pantalla = NULL;
    visible = true;
    habilitado = true;
    clicked = false;
    mouseOver = false;
    TX = 300.0 / zoom;
    TY = 300.0 / zoom;
}

ulong PanelContenedor::getAncho(const string & rutaImg) {
    SDL_Surface * surface = IMG_Load(rutaImg.c_str());
    return (ulong) surface->w;
}

ulong PanelContenedor::getAlto(const string & rutaImg) {
    SDL_Surface * surface = IMG_Load(rutaImg.c_str());
    return (ulong) surface->h;
}

void PanelContenedor::loadTexture(const char * path, int tipoTextura) {
    GLuint textura;
```

```
GLint nOfColors;
GLenum texture_format = 0;

SDL_Surface * surface = IMG_Load(path);

if (surface == NULL) {
    cout << "No se pudo abrir la imagen: " << path << endl;
    return;
}

if ( (surface->w & (surface->w - 1)) != 0) {
    printf("ERROR en imagen, el ancho no es potencia de 2\n");
}

if ( (surface->h & (surface->h - 1)) != 0) {
    printf("ERROR en imagen, el alto no es potencia de 2\n");
}

nOfColors = surface->format->BytesPerPixel;
if (nOfColors == 4)
{
    if (surface->format->Rmask == 0x000000ff)
        texture_format = GL_RGBA;
    else
        texture_format = GL_BGRA;
} else if (nOfColors == 3)
{
    if (surface->format->Rmask == 0x000000ff)
        texture_format = GL_RGB;
    else
        texture_format = GL_BGR;
} else {
    printf("warning: the image is not truecolor.. this will probably break\n");
}

glGenTextures( 1, &textura);

glBindTexture( GL_TEXTURE_2D, textura);

glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

glTexImage2D( GL_TEXTURE_2D, 0, nOfColors, surface->w, surface->h, 0,
texture_format, GL_UNSIGNED_BYTE, surface->pixels);
_texturas.insert(pair<int, GLuint>(tipoTextura, textura));

}

PanelContenedor::~PanelContenedor() {

}

double PanelContenedor::cambioX(long x) {
    return (x - pantalla->w/2.0)/TX;
}

double PanelContenedor::cambioY(long y) {
    return (-y + pantalla->h/2.0)/TY;
}

void PanelContenedor::imprimirImagenDeFondo() {

    glBindTexture( GL_TEXTURE_2D, _texturas[texturaActual]);
    glBegin(GL_QUADS);

    glTexCoord2i( 0, 0);
    glVertex3f(cambioX(posicion.x), cambioY(posicion.y), 0.0f);
    glTexCoord2i( 1, 0);
    glVertex3f(cambioX(posicion.x), cambioY(posicion.y + alto), 0.0f);
    glTexCoord2i( 1, 1);
    glVertex3f(cambioX(posicion.x + ancho), cambioY(posicion.y + alto), 0.0f);
    glTexCoord2i( 0, 1);
    glVertex3f(cambioX(posicion.x + ancho), cambioY(posicion.y), 0.0f);
}
```

```
    glEnd();  
}  
  
void PanelContenedor::imprimir() {  
    imprimirImagenDeFondo();  
    list <PanelContenedor*>::iterator it;  
  
    for (it = _paneles.begin(); it != _paneles.end(); it++) {  
        (*it)->imprimir();  
    }  
}  
  
void PanelContenedor::setAltura(const ulong alto) {  
    this->alto = alto;  
}  
  
void PanelContenedor::setAncho(const ulong ancho) {  
    this->ancho = ancho;  
}  
  
ulong PanelContenedor::getAltura() const {  
    return alto;  
}  
  
ulong PanelContenedor::getAncho() const {  
    return ancho;  
}  
  
void PanelContenedor::setPosicion(const Coordenada posicion) {  
    this->posicion = posicion;  
}  
  
Coordenada PanelContenedor::getPosicion() const {  
    return posicion;  
}  
  
void PanelContenedor::setPantalla(SDL_Surface * pantalla) {  
    this->pantalla = pantalla;  
}  
  
void PanelContenedor::setVentana(Ventana * ventana) {  
    this->ventana = ventana;  
}  
  
string PanelContenedor::getNombre() const {  
    return nombre;  
}  
  
int PanelContenedor::getId() const {  
    return id;  
}  
  
void PanelContenedor::setId(const int id) {  
    this->id = id;  
}  
  
void PanelContenedor::setMouseOver(const bool b) {  
    this->mouseOver = b;  
}  
  
void PanelContenedor::agregarPanel(PanelContenedor * panel,  
                                    Coordenada posicionEnPanel) {  
    panel->setPosicion(this->posicion + posicionEnPanel);  
    panel->setPantalla(pantalla);  
    _paneles.push_back(panel);  
}  
  
void PanelContenedor::agregarPanel(PanelContenedor * panel, int lugar) {  
    Coordenada pos;  
    _paneles.push_back(panel);
```

```
switch (lugar) {
    case 1:
        pos.setX(this->ancho / _paneles.size());
        pos.setY(this->alto + ((this->alto - panel->getAltura()) / 2));
        break;
}

panel->setPantalla(pantalla);
panel->setPosicion(pos);

}

void PanelContenedor::setImagenDeFondo(int imagen) {
    texturaActual = imagen;
}

bool PanelContenedor::pertenece(Coordenada punto) {

    int bordeIzquierdo = posicion.x;
    int bordeDerecho = posicion.x + ancho;
    int bordeSuperior = posicion.y;
    int bordeInferior = posicion.y + alto;

    if ( (punto.x >= bordeIzquierdo) && (punto.x < bordeDerecho) && (punto.y
                                > bordeSuperior) && (punto.y < bordeInferior))
        return true;
    return false;
}

bool PanelContenedor::seEjecutaEventoEn(const Coordenada punto, int evento) {

    if (!pertenece(punto)) {
        return false;
    }

    list<PanelContenedor*>::iterator it;
    for (it = _paneles.begin(); it != _paneles.end(); it++)
        if ((*it)->pertenece(punto))
            if ((*it)->seEjecutaEventoEn(punto, evento))
                return false;
}

if(habilitado) {
    switch (evento) {
        case SDL_MOUSEBUTTONDOWN:
            evento_MouseDown(punto - getPosicion());
            break;
        case SDL_MOUSEMOTION:
            evento_MouseOver(punto - getPosicion());
            break;
        case SDL_MOUSEBUTTONUP:
            evento_MouseUp(punto - getPosicion());
            break;
    }
    return true;
}
return false;
}

void PanelContenedor::evento_MouseDown(Coordenada punto) {
    if (clicked==false) {
        clicked = true;
        if (texturaActual != TEXTURA_MOUSEDOWN)
            texturaActual = TEXTURA_MOUSEDOWN;
        else
            texturaActual = TEXTURA_DEFECTO;
    }
}

void PanelContenedor::evento_MouseOver(Coordenada punto) {

    if (mouseOver == false) {
```

```
        mouseOver = true;
        texturaActual = TEXTURA_MOUSEOVER;
    }

void PanelContenedor::evento_MouseUp(Coordenada punto) {
    if (clicked == true) {
        clicked = false;
    }
}

bool PanelContenedor::is_clicked() const {
    return clicked;
}

bool PanelContenedor::is_over() const {
    return mouseOver;
}

bool PanelContenedor::estaHabilitado() const {
    return habilitado;
}

void PanelContenedor::setHabilitar(const bool b) {
    habilitado = b;
}
```

```
#include "ParserXML.h"

#include <libxml/parser.h>
#include <libxml/tree.h>
#include <iostream>
#include <list>
#include <fstream>
using namespace std;

Mapa* ParserXML::obtenerMapaArchivo (const std::string& rutaEntrada) const
{
    xmlTextReaderPtr reader;
    reader = xmlReaderFromFile(rutaEntrada.c_str(), NULL, 0);
    return obtenerMapa(reader);
}

Mapa* ParserXML::obtenerMapa (xmlTextReaderPtr reader) const
{
    Mapa* mapa = new Mapa();

    int ret;
    if (reader != NULL)
    {
        ret = xmlTextReaderRead(reader);
        while (ret == 1)
        {
            analizarNodo(mapa, reader);
            ret = xmlTextReaderRead(reader);
        }
    }
    mapa->actualizarEnlaces();
    mapa->setNroElementoParaCrear(NINGUN_ELEMENTO);
    return mapa;
}

int ParserXML::nroElemento(const std::string& tipo) const
{
    string nom = tipo.substr(tipo.find_last_of(":")+1,tipo.length()-tipo.find_last_of(":")-1);

    if (nom.compare("Masa") == 0)
        return MASA;

    if (nom.compare("Rueda") == 0)
        return RUEDA;

    if (nom.compare("PuntoFijo") == 0)
        return PUNTO_FIJO;

    if (nom.compare("Barra") == 0)
        return BARRA;

    if (nom.compare("Plataforma") == 0)
        return PLATAFORMA;

    if (nom.compare("Soga") == 0)
        return SOGA;

    if (nom.compare("Cinta") == 0)
        return CINTA;

    if (nom.compare("Cohete") == 0)
        return COHETE;

    if (nom.compare("ZonaLlegada") == 0)
        return ZONA_LLEGADA;

    return NINGUN_ELEMENTO;
}
```

```
void ParserXML::procesarElemento(Mapa* mapa, xmlTextReaderPtr reader) const
{
    const xmlChar *value;
    string tipo;
    int nroTipo;
    char* tipoAux;
    long angulo;
    ulong magnitud;
    Coordenada coor;

    xmlTextReaderRead(reader);

    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader); //leo el tipo
    value = xmlTextReaderConstValue(reader);
        tipoAux = (char*) value;
        tipo = tipoAux;
        nroTipo = nroElemento(tipo);
    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader);

    mapa->setNroElementoParaCrear(nroTipo);

    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader); //leo la magnitud
    value = xmlTextReaderConstValue(reader);
        magnitud = atoi ((const char*) value);
    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader);

    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader); //leo el angulo
    value = xmlTextReaderConstValue(reader);
        angulo = atoi ((const char*) value);
    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader);

    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader); //leo la posicion

    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader); //leo x
    value = xmlTextReaderConstValue(reader);
        coor.x = atoi ((const char*) value);
    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader);

    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader); //leo y
    value = xmlTextReaderConstValue(reader);
        coor.y = atoi ((const char*) value);
    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader);

    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader);

    mapa->agregarElemento(coor);

    if ((nroTipo >= BARRA) && (nroTipo <= COHETE))
    {
        Coordenada extremoOpuesto;

        xmlTextReaderRead(reader);
        xmlTextReaderRead(reader);

        xmlTextReaderRead(reader);
        xmlTextReaderRead(reader);
        //leo x
        value = xmlTextReaderConstValue(reader);
```

```
extremoOpuesto.x = atoi((const char*) value);
xmlTextReaderRead(reader);
xmlTextReaderRead(reader);

xmlTextReaderRead(reader);
xmlTextReaderRead(reader);
//leo y
value = xmlTextReaderConstValue(reader);
extremoOpuesto.y = atoi((const char*) value);
xmlTextReaderRead(reader);
xmlTextReaderRead(reader);

xmlTextReaderRead(reader);
xmlTextReaderRead(reader);

mapa->setExtremoOpuesto(extremoOpuesto);

}

mapa->agrandarElementoA(magnitud);
mapa->rotarElementoA(angulo);

// xmlTextReaderRead(reader);
// xmlTextReaderRead(reader);

xmlTextReaderRead(reader);

}

void ParserXML::procesarHabilitados(Mapa* mapa, xmlTextReaderPtr reader) const
{
    const xmlChar *value;
    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader);
    value = xmlTextReaderConstValue(reader); //leo masa
    if (strcmp((const char*)value,"NO") == 0)
        mapa->deshabilitarElemento(MASA);
    else
        mapa->habilitarElemento(MASA);
    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader);

    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader);
    value = xmlTextReaderConstValue(reader); //leo punto fijo
    if (strcmp((const char*)value,"NO") == 0)
        mapa->deshabilitarElemento(PUNTO_FIJO);
    else
        mapa->habilitarElemento(PUNTO_FIJO);
    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader);

    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader);
    value = xmlTextReaderConstValue(reader); //leo rueda
    if (strcmp((const char*)value,"NO") == 0)
        mapa->deshabilitarElemento(RUEDA);
    else
        mapa->habilitarElemento(RUEDA);
    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader);

    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader);
    value = xmlTextReaderConstValue(reader); //leo barra
    if (strcmp((const char*)value,"NO") == 0)
        mapa->deshabilitarElemento(BARRA);
```

```
else
    mapa->habilitarElemento(BARRA);
xmlTextReaderRead(reader);
xmlTextReaderRead(reader);

xmlTextReaderRead(reader);
xmlTextReaderRead(reader);
value = xmlTextReaderConstValue(reader); //leo plataforma
if (strcmp((const char*)value,"NO") == 0)
    mapa->deshabilitarElemento(PLATAFORMA);
else
    mapa->habilitarElemento(PLATAFORMA);
xmlTextReaderRead(reader);
xmlTextReaderRead(reader);

xmlTextReaderRead(reader);
xmlTextReaderRead(reader);
value = xmlTextReaderConstValue(reader); //leo soga
if (strcmp((const char*)value,"NO") == 0)
    mapa->deshabilitarElemento(SOGA);
else
    mapa->habilitarElemento(SOGA);
xmlTextReaderRead(reader);
xmlTextReaderRead(reader);

xmlTextReaderRead(reader);
xmlTextReaderRead(reader);
value = xmlTextReaderConstValue(reader); //leo cinta
if (strcmp((const char*)value,"NO") == 0)
    mapa->deshabilitarElemento(CINTA);
else
    mapa->habilitarElemento(CINTA);
xmlTextReaderRead(reader);
xmlTextReaderRead(reader);

xmlTextReaderRead(reader);
xmlTextReaderRead(reader);
value = xmlTextReaderConstValue(reader); //leo cohete
if (strcmp((const char*)value,"NO") == 0)
    mapa->deshabilitarElemento(COHETE);
else
    mapa->habilitarElemento(COHETE);
xmlTextReaderRead(reader);
xmlTextReaderRead(reader);

xmlTextReaderRead(reader);

}

void ParserXML::procesarCostos(Mapa* mapa, xmlTextReaderPtr reader) const
{
    const xmlChar *value;
    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader);
    value = xmlTextReaderConstValue(reader); //leo masa
    mapa->setPrecio(MASA,atoi((const char*)value));
    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader);

    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader);
    value = xmlTextReaderConstValue(reader); //leo punto fijo
    mapa->setPrecio(PUNTO_FIJO,atoi((const char*)value));

    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader);

    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader);
}
```

```
    value = xmlTextReaderConstValue(reader); //leo rueda
    mapa->setPrecio(RUEDA,atoi((const char*)value));
    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader);

    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader);
    value = xmlTextReaderConstValue(reader); //leo barra

    mapa->setPrecio(BARRA,atoi((const char*)value));
    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader);

    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader);
    value = xmlTextReaderConstValue(reader); //leo plataforma

    mapa->setPrecio(PLATAFORMA,atoi((const char*)value));
    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader);

    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader);
    value = xmlTextReaderConstValue(reader); //leo soga

    mapa->setPrecio(SOGA,atoi((const char*)value));
    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader);

    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader);
    value = xmlTextReaderConstValue(reader); //leo cinta

    mapa->setPrecio(CINTA,atoi((const char*)value));
    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader);

    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader);
    value = xmlTextReaderConstValue(reader); //leo cohete

    mapa->setPrecio(COHETE,atoi((const char*)value));
    xmlTextReaderRead(reader);
    xmlTextReaderRead(reader);

    xmlTextReaderRead(reader);
}

}
```

```
void ParserXML::analizarNodo(Mapa* mapa, xmlTextReaderPtr reader) const
{
    const xmlChar *name, *value;
    char* valueAux;

    name = xmlTextReaderConstName(reader);

    if (strcmp((const char*)name,"nombre") == 0)
    {
        xmlTextReaderRead(reader);
        value = xmlTextReaderConstValue(reader);
        valueAux = (char*) value;
        mapa->setNombre(valueAux);//asignar el nombre al mapa
        xmlTextReaderRead(reader);
        xmlTextReaderRead(reader);
    }
    else if (strcmp((const char*)name,"plata") == 0)
    {
        xmlTextReaderRead(reader);
    }
}
```

```

        value = xmlTextReaderConstValue(reader);
        valueAux = (char*) value;
        mapa->setPlata(atoi(valueAux));
        xmlTextReaderRead(reader);
    }
    else if (strcmp((const char*)name, "elemento") == 0)
        procesarElemento(mapa, reader);

    else if (strcmp((const char*)name, "habilitados") == 0)
        procesarHabilitados(mapa, reader);

    else if (strcmp((const char*)name, "costes") == 0)
        procesarCostos(mapa, reader);

    else if (strcmp((const char*)name, "TiempoCohete") == 0)
    {
        xmlTextReaderRead(reader);
        value = xmlTextReaderConstValue(reader);
        valueAux = (char*) value;
        mapa->setTiempoCohete(atoi(valueAux));
        xmlTextReaderRead(reader);
    }
}

Mapa* ParserXML::obtenerMapaMemoria (const std::string& xml) const
{
    xmlTextReaderPtr reader;

    reader = xmlReaderForMemory(xml.c_str(), xml.length(), NULL, NULL, 0);

    return obtenerMapa(reader);
}

void ParserXML::obtenerXMLArchivo (const std::string& rutaSalida, const Mapa* mapa) const
{
    string xml;
    obtenerXMLMemoria(xml, mapa);
    ofstream arch(rutaSalida.c_str());
    arch.write(xml.c_str(),xml.length());
    arch.close();
}

void ParserXML::enteroAString(std::string& cadena, ulong entero) const
{
    if (entero == 0)
        cadena = "0";
    else
        cadena = "";
    while (entero != 0)
    {
        switch (entero %10)
        {
            case 0: cadena="0"+cadena; break;
            case 1: cadena="1"+cadena; break;
            case 2: cadena="2"+cadena; break;
            case 3: cadena="3"+cadena; break;
            case 4: cadena="4"+cadena; break;
            case 5: cadena="5"+cadena; break;
            case 6: cadena="6"+cadena; break;
            case 7: cadena="7"+cadena; break;
            case 8: cadena="8"+cadena; break;
            case 9: cadena="9"+cadena; break;
        }
        entero /= 10;
    }
}

void ParserXML::imprimirElementos(std::string& xml, const Mapa* mapa) const

```

```
{  
    list<Elemento*> lista = mapa->getListaElementos();  
    Elemento* actual;  
    string nro;  
    Coordenada coorActual;  
    list<Elemento*>::iterator it = lista.begin();  
    while (it != lista.end())  
    {  
        actual = *it;  
        xml += " ";  
        xml += "<elemento>\n";  
  
        xml += " " <tip>;  
        xml += actual->getNombre();  
        xml += "</tip>\n";  
  
        xml += " " <magnitud>;  
        enteroAString(nro,actual->getMagnitud());  
        xml += nro;  
        xml += "</magnitud>\n";  
  
        xml += " " <angulo>;  
        if ((actual->getAngulo() < 0)  
            enteroAString(nro,actual->getAngulo()+360);  
        else  
            enteroAString(nro,actual->getAngulo());  
        xml += nro;  
        xml += "</angulo>\n";  
  
        coorActual = actual->getPosicion();  
  
        xml += " " <posicion>\n";  
  
        xml += " " <x>;  
        enteroAString(nro,coorActual.x);  
        xml += nro;  
        xml += "</x>\n";  
  
        xml += " " <y>;  
        enteroAString(nro,coorActual.y);  
        xml += nro;  
        xml += "</y>\n";  
  
        xml += " " </posicion>\n";  
  
        if (actual->es(LONGITUDINAL))  
        {  
            coorActual = actual->getExtremoOpuesto();  
  
            xml += " " <extremo_opuesto>\n";  
  
            xml += " " <x>;  
            enteroAString(nro,coorActual.x);  
            xml += nro;  
            xml += "</x>\n";  
  
            xml += " " <y>;  
            enteroAString(nro,coorActual.y);  
            xml += nro;  
            xml += "</y>\n";  
  
            xml += " " </extremo_opuesto>\n";  
        }  
  
        xml += " ";  
        xml += "</elemento>\n";  
    }  
}
```

```
        it++;
    }

}

void ParserXML::obtenerXMLMemoria (const Mapa* mapa) const
{
    string plata;
    enteroAString(plata,mapa->getPlata());

    xml.clear();

/*
    xml += "<?xml version=";
    xml += "'";
    xml += "1.0";
    xml += "'";
    xml += " encoding=";
    xml += "'";
    xml += "UTF-8";
    xml += "'";
    xml += "?>\n"; */

    xml += "<mapa>\n";
    xml += "      ";
    xml += "<nombre>";
    xml += mapa->getNombre();
    xml += "</nombre>\n";
    xml += "      ";
    xml += "<plata>";
    xml += plata;
    xml += "</plata>\n";
    xml += "      ";
    xml += "<elementos>\n";
    imprimirElementos(xml,mapa);
    xml += "</elementos>\n";
    imprimirHabilitados(xml,mapa);
    imprimirCostes(xml,mapa);
    imprimirTiempo(xml,mapa);
    xml += "      ";
    xml += "</mapa>";
}

void ParserXML::imprimirHabilitados(const Mapa* mapa) const
{
    xml += "      <habilitados>\n";

    xml += "          <Masa>";
    if (mapa->elementoHabilitado(MASA))
        xml += "SI";
    else
        xml += "NO";
    xml += "</Masa>\n";

    xml += "          <Punto_Fijo>";
    if (mapa->elementoHabilitado(PUNTO_FIJO))
        xml += "SI";
    else
        xml += "NO";
    xml += "</Punto_Fijo>\n";

    xml += "          <Rueda>";
    if (mapa->elementoHabilitado(RUEDA))
        xml += "SI";
    else
        xml += "NO";
    xml += "</Rueda>\n";

    xml += "          <Barra>";
    if (mapa->elementoHabilitado(BARRA))
        xml += "SI";
    else
```

```
        xml += "NO";
xml += "</Barra>\n";

xml += "                <Plataforma>";
if (mapa->elementoHabilitado(PLATAFORMA))
    xml += "SI";
else
    xml += "NO";
xml += "</Plataforma>\n";

xml += "                <Soga>";
if (mapa->elementoHabilitado(SOGA))
    xml += "SI";
else
    xml += "NO";
xml += "</Soga>\n";

xml += "                <Cinta>";
if (mapa->elementoHabilitado(CINTA))
    xml += "SI";
else
    xml += "NO";
xml += "</Cinta>\n";

xml += "                <Cohete>";
if (mapa->elementoHabilitado(COHETE))
    xml += "SI";
else
    xml += "NO";
xml += "</Cohete>\n";

xml += "            </habilitados>\n";
}

void ParserXML::imprimirCostes(std::string& xml, const Mapa* mapa) const
{
    string nro;

    xml += "            <costes>\n";

    xml += "                <Masa>";
    enteroAString(nro, mapa->getPrecio(MASA));
    xml += nro;
    xml += "</Masa>\n";

    xml += "                <Punto_Fijo>";
    enteroAString(nro, mapa->getPrecio(PUNTO_FIJO));
    xml += nro;
    xml += "</Punto_Fijo>\n";

    xml += "                <Rueda>";
    enteroAString(nro, mapa->getPrecio(RUEDA));
    xml += nro;
    xml += "</Rueda>\n";

    xml += "                <Barra>";
    enteroAString(nro, mapa->getPrecio(BARRA));
    xml += nro;
    xml += "</Barra>\n";

    xml += "                <Plataforma>";
    enteroAString(nro, mapa->getPrecio(PLATAFORMA));
    xml += nro;
    xml += "</Plataforma>\n";

    xml += "                <Soga>";
    enteroAString(nro, mapa->getPrecio(SOGA));
    xml += nro;
    xml += "</Soga>\n";

    xml += "                <Cinta>";
```

```
    enteroAString(nro, mapa->getPrecio(CINTA));
    xml += nro;
    xml += "</Cinta>\n";

    xml += "          <Cohete>";
    enteroAString(nro, mapa->getPrecio(COHETE));
    xml += nro;
    xml += "</Cohete>\n";

    xml += "      </costes>\n";
}

void ParserXML::imprimirTiempo(std::string& xml, const Mapa* mapa) const
{
    string nro;
    enteroAString(nro, mapa->getTiempoCohete());
    xml += "          <TiempoCohete>";
    xml += nro;
    xml += "</TiempoCohete>\n";
}
```

```
#include "Plataforma.h"

Plataforma::Plataforma() :
    ElementoLongitudinal("Longitudinal:Plataforma", Coordenada(long(10), long(10)), 25, 0, 10) {}

Plataforma::Plataforma(Cordenada posicion, ulong magnitud, long angulo, long ancho) :
    ElementoLongitudinal("Longitudinal:Plataforma", posicion, magnitud, angulo, ancho) {}

Plataforma::~Plataforma() {
```

```
#include "PuntoDeEnlace.h"

PuntoDeEnlace::PuntoDeEnlace() :
    ElementoEnlazable("Circular:Enlazable:PuntoDeEnlace", Coordenada(long(100), long(100)), 15, 0,
NULL) {}

PuntoDeEnlace::PuntoDeEnlace(Coordenada posicion, Elemento * poseedor) :
    ElementoEnlazable("Circular:Enlazable:PuntoDeEnlace", posicion, 10, 0, poseedor) {}

PuntoDeEnlace::~PuntoDeEnlace() {
```

```
#include "PuntoFijo.h"

PuntoFijo::PuntoFijo() :
    ElementoEnlazable("Circular:Enlazable:PuntoFijo", Coordenada(long(100), long(100)), 10, 0,
NULL) {}

PuntoFijo::PuntoFijo(Coordenada posicion, ulong magnitud) :
    ElementoEnlazable("Circular:Enlazable:PuntoFijo", posicion, magnitud, 0, NULL) {}

PuntoFijo::~PuntoFijo() {}
```

```
#include "Reloj.h"
#include <iostream>
using namespace std;

#define TAM_LETRA_X 0.05
#define TAM_LETRA_Y 0.1

void Reloj::cargarTexturas()
{
    string path;

    char i;
/*    for (i = 'a' ; i <= 'z' ; i++)
    {
        path = "letras/letra__";
        path += i;
        path += ".bmp";
        loadTexture(path.c_str(),i);
    }*/
    for (i = '0' ; i <= '9' ; i++)
    {
        path = "letras/letra__";
        path += i;
        path += ".bmp";
        loadTexture(path.c_str(),i);
    }
    loadTexture("letras/letra_s.bmp",'s');
    loadTexture("letras/vacio.bmp",' ');
    loadTexture("letras/letra_dos_puntos.bmp",';');
    loadTexture("letras/letra_punto.bmp",'.');
//    loadTexture("letras/letra_coma.bmp",';');
//    loadTexture("letras/letra_admiracion.bmp",'!');
//    loadTexture("letras/letra_interrogacion.bmp",'?');
//    loadTexture("letras/cursor.bmp",'=');
}

void Reloj::imprimirReloj(const std::string& reloj, float x, float y)
{
    unsigned int i;

    for ( i = 0 ; i < reloj.length() ; i++)
    {
        imprimirCaracter(reloj[i],x,y);
        x += TAM_LETRA_X;
    }

}

void Reloj::imprimirCaracter(char letra, float x, float y)
{
    map<char,GLuint>::iterator it = texturas.find(letra);
    if (it == texturas.end())
        return;

    GLuint textura = it->second;

    glBindTexture( GL_TEXTURE_2D, textura);

    glBegin(GL_QUADS);
    {
        glTexCoord2i( 0, 0 );
        glVertex3f( x, y, 0.0f );
        glTexCoord2i( 0, 1 );
        glVertex3f( x, y - TAM_LETRA_Y, 0.0f );
        glTexCoord2i( 1, 1 );
        glVertex3f( x + TAM_LETRA_X, y - TAM_LETRA_Y, 0.0f );
        glTexCoord2i( 1, 0 );
        glVertex3f( x + TAM_LETRA_X, y, 0.0f );
    }
    glEnd();
}
```

}

```
void Reloj::loadTexture(const char * path, char letra) {
    GLuint texture;
    GLint nOfColors;
    GLenum texture_format = 0;
    SDL_Surface *surface;

    if ( (surface = SDL_LoadBMP(path)) ) {

        // Check that the image's width is a power of 2
        if ( (surface->w & (surface->w - 1)) != 0 ) {
            printf("ERROR en imagen, el ancho no es potencia de 2\n");
        }

        // Also check if the height is a power of 2
        if ( (surface->h & (surface->h - 1)) != 0 ) {
            printf("ERROR en imagen, el alto no es potencia de 2\n");
        }

        // get the number of channels in the SDL surface
        nOfColors = surface->format->BytesPerPixel;
        if (nOfColors == 4)      // contains an alpha channel
        {
            if (surface->format->Rmask == 0x000000ff)
                texture_format = GL_RGBA;
            else
                texture_format = GL_BGRA;
        } else if (nOfColors == 3) // no alpha channel
        {
            if (surface->format->Rmask == 0x000000ff)
                texture_format = GL_RGB;
            else
                texture_format = GL_BGR;
        } else {
            printf("warning: the image is not truecolor.. this will probably break\n");
            // this error should not go unhandled
        }
    }

    // Have OpenGL generate a texture object handle for us
    glGenTextures( 1, &texture );

    // Bind the texture object
    glBindTexture( GL_TEXTURE_2D, texture );

    // Set the texture's stretching properties
    glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR );
    glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR );

    // Edit the texture object's image data using the information SDL_Surface gives us
    glTexImage2D( GL_TEXTURE_2D, 0, nOfColors, surface->w, surface->h, 0,
                  texture_format, GL_UNSIGNED_BYTE, surface->pixels );
    texturas.insert(make_pair(letra, texture));
}

else {
    printf("SDL could not load image.bmp: %s\n", SDL_GetError());
    SDL_Quit();
}
```

```
#include "Rueda.h"
#include <math.h>
using namespace std;

Rueda::Rueda() :
    ElementoCircular("Circular:Rueda", Coordenada(long(100), long(100)), 10, 0) {
}

Rueda::Rueda(Coordenada posicion, ulong magnitud) :
    ElementoCircular("Circular:Rueda", posicion, magnitud, 0) {
}

Rueda::~Rueda() {
    eliminarEnlaces();
    _enlaces.clear();
}

void Rueda::actualizarPuntosDeEnlace() {
    eliminarEnlaces();
    _enlaces.clear();

    ulong radio = getMagnitud();
    ulong x = getPosition().x;
    ulong y = getPosition().y;

    for (int j = 0; j <= POLY_SIDES; j += 2) {
        Coordenada pos;
        pos.setX(x + radio * cos(j * (2 * PI
                                         / POLY_SIDES)));
        pos.setY(y + radio * sin(j * (2 * PI
                                         / POLY_SIDES)));

        _enlaces.push_back(new PuntoDeEnlace(pos, this));
    }
}

void Rueda::eliminarEnlaces() {
    list <PuntoDeEnlace*>::iterator it;
    for(it = _enlaces.begin(); it != _enlaces.end(); it++) {
        delete (*it);
    }
    _enlaces.clear();
}

void Rueda::imprimirEnlaces() {
    list <PuntoDeEnlace*>::iterator it;
    cout << "Cantidad de enlaces: " << _enlaces.size() << endl;
    for(it = _enlaces.begin(); it != _enlaces.end(); it++) {
        (*it)->getPosition().imprimir();
    }
    cout << endl << endl;
}

std::list<PuntoDeEnlace*> Rueda::getListaEnlaces() {
    return _enlaces;
}
```

```
#include "sms_app.h"
#include "loader.h"
#include <cmath>
#include <fstream>
#include <SDL.h>
#include <SDL_opengl.h>
#include <sstream>
#include <vector>
#include <iostream>
#include "Angulo.h"
#include "constantes.h"
#include "constantesnico.h"
#include "hiloCron.h"
#define SHOW_ALL 0
#define SHOW_TEXT 1
#define ANCHO 0.02

using namespace std;
namespace
{
    const double MASS_SIZE = 0.02;
    const int POLY_SIDE = 12;

    const double MAX_DT = 0.003;

    const std::string CONFIG_FILENAME = "config.txt";
}

SMSApp::SMSApp(int argc, Mapa* mapa) :
config_(CONFIG_FILENAME),
sms_(config_),
video_(config_),
ticks_(0),
physMult_(config_.read<int>("App.PhysMultiplier"))
{
    controlLonas=0;
    reloj.cargarTexturas();
    tiempoLlegada = NULL;
    estabaEnLlegada = false;
    sms = false;
    sms_.setMapa(mapa);
    if (argc == 0)
        load(sms_);
    else
    {
        sms=true;
        load(sms_);
    }
}

SMSApp::~SMSApp()
{
}

bool SMSApp::estaEnLlegada(){
    if ((sms_.getElementManager())->getMainBalls().size()){
        MassProxy masa = sms_.getMassProxy((sms_.getElementManager())->getMainBalls().at(0)->getFirst());
        Vector2<> m = masa.getPos();
        double rad = RADIOMASA;
        vector<ZonaLlegada*> llegadas = (sms_.getElementManager())->getZonasLlegada();
        ZonaLlegada* z;

        for (size_t i= 0 ; i<llegadas.size(); i++)
        {
            z = llegadas.at(i);
            if (
                ((m.x - rad) > z->getX()) &&
                ((m.x + rad) < (z->getX() + z->getWidth()) ) &&
                ((m.y + rad) < z->getY()) &&
                ((m.y - rad) > z->getY())
            )
                return true;
        }
    }
    return false;
}
```

```
        ((m.y - rad) > (z->getY() - z->getHeigth() ) )
    )
    return true;
}
}
return false;
}

bool SMSApp::ganaNivel(){

    if (estaEnLlegada()){
        if (!estabaEnLlegada){
            tiempoLlegada = new HiloCron();
            tiempoLlegada->start();
            estabaEnLlegada = true;
        }
    else
        if (tiempoLlegada->getMilisegundos() > TIEMPOGANAR)
            return true;
    }
else {
    estabaEnLlegada = false;
    if (tiempoLlegada){
        tiempoLlegada->terminar();
        tiempoLlegada->join();
        delete(tiempoLlegada);
        tiempoLlegada = NULL;
    }
}
return false;
}

int SMSApp::run()
{
    bool gano = false;
    HiloCron cronometro;
    bool exit_now = false;

    ticks_ = SDL_GetTicks();

    SDL_Event ev;

    cronometro.start();
    while (!exit_now)
    {
        while (SDL_PollEvent(&ev)){
            if (ev.type == SDL_QUIT)
                exit_now = true;
            if (ev.type == SDL_KEYDOWN)
                if ( ev.key.keysym.sym == SDLK_ESCAPE )
                    exit_now = true;
                else if (!strcmp(SDL_GetKeyName(ev.key.keysym.sym), "s"))
                    sms = !sms;
        }
        uint32_t newTicks = SDL_GetTicks();
        double dt;

        dt = MAX_DT;

        sms_.rk4Step(dt);
        ticks_ = newTicks;

        /*****IMPRIMIR RELOJ*****/
        double tiempo = cronometro.getMilisegundos() / TIME_C;
        stringstream ss;

        ss << fixed;
        ss.precision( 2 );
        ss << tiempo;

        /*****FIN IMPRIMIR RELOJ*****/
    }
}
```

```
        string time = ss.str();
        time += " s.";
        drawFrame(time);
        if (gano = ganaNivel())
            exit_now = true;

    }

double tiempo = cronometro.getMilisegundos() / TIME_C;
cronometro.terminar();
cronometro.join();
stringstream ss;
ss << tiempo;
if (gano)
    return 0;
return -1;
}

bool SMSApp::erased(int spring){
    map<int,int>::iterator it;
    if (borrados.find(spring)==borrados.end()){
        borrados.insert(make_pair(spring,0));
        return false;
    }
    else return true;
}
void SMSApp::drawFrame(string &time)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    int status;
    status=SHOW_ALL;
    if (!sms)
        status=SHOW_TEXT;

    reloj.imprimirReloj(time, CLOCK_X, CLOCK_Y);
    // draws all springs
    glBegin(GL_LINES);
    if (status==SHOW_ALL)
        for (size_t i = 0; i < sms_.getNumSprings(); i++)
    {
        SpringProxy sp = sms_.getSpringProxy(i);

        Vector2<> startPos = sp.getStartPos();
        Vector2<> endPos = sp.getEndPos();

        double relElong = sp.getRelElongation();
        double k = sp.getK();

        if (k == 0.0)
            continue;

        double color = relElong * k / 10.0;
        if (color < -1.0)
            color = -1.0;
        else if (color > 1.0)
            color = 1.0;

        if (color < 0.0)
        {
            glColor3d(-color / 2.0 + 0.5, 0.5 + color / 2.0,
                      0.5 + color / 2.0);
        }
        else
        {
            double green = 1.0 - 1.0 / (2.0 + 2.0 * color);
            glColor3d(1.0 - green, green, 1.0 - green);
        }

        glVertex2d(startPos.x, startPos.y);
        glVertex2d(endPos.x, endPos.y);
    }
    glEnd();
}
```

```

// draws all masses
if (status==SHOW_ALL)
for (size_t i = 0; i < sms_.getNumMasses(); i++)
{
    MassProxy mp = sms_.getMassProxy(i);
    Vector2<> massPos = mp.getPos();
    double invMass = mp.getInvMass();
    double massSize;
    if (invMass == 0.0)
        massSize = MASS_SIZE;
    else
        massSize = MASS_SIZE / sqrt(invMass);
    glBegin(GL_TRIANGLE_FAN);
        if (invMass != 0.0)
            glColor3f(0.4,0.5, 1);
        else
            glColor3f(0, 0.1, 0.5);
    glVertex2d(massPos.x, massPos.y);
    for (int j = 0; j <= POLY_SIDE; j++)
        glVertex2d(
            massPos.x + 0.007 * cos(j * (2 * PI / POLY_SIDE)),
            massPos.y + 0.007 * sin(j * (2 * PI / POLY_SIDE)));
    glEnd();
}

glColor4d(1.0, 1.0, 1.0, 1.0);
unsigned it;

//IMPRIME LAS LONAS
vector<Element*> lonas = (sms_.getElementManager())->getLonas();

std::vector<Element*>::iterator i = lonas.begin();
std::map<int,int> springs =(sms_.getElementManager())->getSpringElements();
std::map<int,int>::iterator ite;
for (ite=springs.begin(); ite!=springs.end();ite++){
    SpringProxy sp = sms_.getSpringProxy(ite->first);
    if (!sp.getK() && !erased(ite->first)){
        lonas.erase(i+ite->second);
        controlLonas++;
    }
}
glBindTexture( GL_TEXTURE_2D, video_.textures.at(CINTA) );
glBegin(GL_QUADS);
if (status==SHOW_TEXT)
for (it=0; it<lonas.size();it++){ //para cada una de las lonas
    for (size_t i = 0; i < lonas.at(it)->masas.size()-1; i++)
    {
        MassProxy mp = sms_.getMassProxy(lonas.at(it)->masas.at(i));
        Vector2<> massPos = mp.getPos();
        MassProxy mp2 = sms_.getMassProxy(lonas.at(it)->masas.at(i+1));
        Vector2<> massPos2 = mp2.getPos();
        double ang;
        double lseg = sqrt( pow(massPos.x-massPos2.x,2) + pow(massPos.y-
massPos2.y,2) );
        double dx,dy;
        ang = atan(ANCHO / lseg);
        dx = sin (ang) * ANCHO;
        dy = cos (ang) * ANCHO;

        glTexCoord2i( 0, 0 );
        glVertex3f( massPos.x, massPos.y, 0.0f );
        glTexCoord2i( 1, 0 );
        glVertex3f( massPos.x + dx, massPos.y - dy, 0.0f );
        glTexCoord2i( 1, 1 );
        glVertex3f( massPos2.x + dx, massPos2.y - dy, 0.0f );
        glTexCoord2i( 0, 1 );
        glVertex3f( massPos2.x, massPos2.y, 0.0f );
}
}

```

```
        }
    }
glEnd();

//IMPRIME LAS BARRAS DE METAL
vector<Element*> metalBars = (sms_.getElementManager())->getMetalBars();
glBindTexture( GL_TEXTURE_2D, video_.textures.at(BARRA) );
glBegin(GL_QUADS);
if (status==SHOW_TEXT)
for (it=0; it<metalBars.size();it++){
for (size_t i = 0; i < (metalBars.at(it)->masas.size())/2-1; i++)
{
    MassProxy mp = sms_.getMassProxy(metalBars.at(it)->masas.at(i));
    Vector2<> massPos = mp.getPos();
    MassProxy mp2 = sms_.getMassProxy(metalBars.at(it)->masas.at(i+1));
    Vector2<> massPos2 = mp2.getPos();
    double ang;
    double lseg = sqrt( pow(massPos.x-massPos2.x,2) + pow(massPos.y-
massPos2.y,2) );
    double dx,dy;
    ang = atan(ANCHO / lseg);
    dx = sin (ang) * ANCHO;
    dy = cos (ang) * ANCHO;

    glTexCoord2i( 0, 0 );
    glVertex3f( massPos.x, massPos.y, 0.0f );
    glTexCoord2i( 1, 0 );
    glVertex3f( massPos.x + dx, massPos.y - dy, 0.0f );
    glTexCoord2i( 1, 1 );
    glVertex3f( massPos2.x+ dx, massPos2.y - dy, 0.0f );
    glTexCoord2i( 0, 1 );
    glVertex3f( massPos2.x, massPos2.y, 0.0f );
}
}
glEnd();

//IMPRIME LAS PLATAFORMAS
vector<Element*> plataformas = (sms_.getElementManager())->getPlataformas();

glBindTexture( GL_TEXTURE_2D, video_.textures.at(PLATAFORMA) );
glBegin(GL_QUADS);
if (status==SHOW_TEXT)
for (it=0; it<plataformas.size();it++){
for (size_t i = 0; i < (plataformas.at(it)->masas.size())/2-1; i++)
{
    MassProxy mp = sms_.getMassProxy(plataformas.at(it)->masas.at(i));
    Vector2<> massPos = mp.getPos();
    MassProxy mp2 = sms_.getMassProxy(plataformas.at(it)->masas.at(i+1));
    Vector2<> massPos2 = mp2.getPos();
    double ang;
    double lseg = sqrt( pow(massPos.x-massPos2.x,2) + pow(massPos.y-
massPos2.y,2) );
    ang = atan( (massPos2.y - massPos.y) / (massPos2.x - massPos.x) );
    double dx,dy;
    ang += PI / 2;
    dx = sin (Angulo::aRadianes(ang)) * ANCHO;
    dy = cos (Angulo::aRadianes(ang)) * ANCHO;

    glTexCoord2i( 0, 0 );
    glVertex3f( massPos.x, massPos.y, 0.0f );
    glTexCoord2i( 1, 0 );
    glVertex3f( massPos.x + dx, massPos.y + dy, 0.0f );
    glTexCoord2i( 1, 1 );
    glVertex3f( massPos2.x+ dx, massPos2.y + dy, 0.0f );
    glTexCoord2i( 0, 1 );
    glVertex3f( massPos2.x, massPos2.y, 0.0f );

    /*massPos.x += ANCHO/2* cos(ang );
    massPos.y -= ANCHO/2* sin(ang );
```

```
        double aX = massPos.x - ANCHO/2 * cos(ang );
        double aY = massPos.y + ANCHO/2 * sin(ang );
        double bX = (lseg)
                    *cos(ang) + aX;
        double bY = (lseg)
                    *sin(ang) + aY;
        double cX = (ANCHO)*sin(ang)
                    + aX;
        double cY = aY - (ANCHO)*cos(ang);
        double dX = cX + (lseg)
                    *cos(ang);
        double dY = cY + (lseg)
                    *sin(ang);

        glTexCoord2i( 0, 0 );
        glVertex3f(cX, cY, 0.0f);
        glTexCoord2i( 1, 0 );
        glVertex3f(aX, aY, 0.0f);
        glTexCoord2i( 1, 1 );
        glVertex3f(bX, bY, 0.0f);
        glTexCoord2i( 0, 1 );
        glVertex3f(dX, dY, 0.0f); */

    }
}

//IMPRIME LAS SOGAS
vector<Element*> ropes = (sms_.getElementManager())->getRopes();

glBindTexture( GL_TEXTURE_2D, video_.textures.at(SOGA) );
glBegin(GL_QUADS);
if (status==SHOW_TEXT)
for (it=0; it<ropes.size();it++){ //para cada una de las sogas
    for (size_t i = 0; i < ropes.at(it)->masas.size()-1; i++)
    {
        MassProxy mp = sms_.getMassProxy(ropes.at(it)->masas.at(i));
        Vector2<> massPos = mp.getPos();
        MassProxy mp2 = sms_.getMassProxy(ropes.at(it)->masas.at(i+1));
        Vector2<> massPos2 = mp2.getPos();

        if ( abs(massPos.x - massPos2.x) < 0.009){
            glTexCoord2i( 0, 0 );
            glVertex3f( massPos.x, massPos.y, 0.0f );
            glTexCoord2i( 1, 0 );
            glVertex3f( massPos2.x, massPos2.y-0.02, 0.0f );
            glTexCoord2i( 1, 1 );
            glVertex3f( massPos2.x+0.005, massPos2.y, 0.0f );
            glTexCoord2i( 0, 1 );
            glVertex3f( massPos.x+0.005, massPos.y, 0.0f );
        }
        else {
            glTexCoord2i( 0, 0 );
            glVertex3f( massPos.x, massPos.y, 0.0f );
            glTexCoord2i( 1, 0 );
            glVertex3f( massPos.x, massPos.y-0.02, 0.0f );
            glTexCoord2i( 1, 1 );
            glVertex3f( massPos2.x, massPos2.y-0.02, 0.0f );
            glTexCoord2i( 0, 1 );
            glVertex3f( massPos2.x, massPos2.y, 0.0f );
        }
    }
}
glEnd();

//IMPRIME LAS BOLAS
vector<Element*> balls = (sms_.getElementManager())->getMainBalls();

if (status==SHOW_TEXT)
```

```

    for (it=0; it<balls.size();it++){
        size_t cant = balls.at(it)->getLast() - balls.at(it)->getFirst
    ();
        MassProxy central = sms_.getMassProxy(balls.at(it)->getFirst());
        Vector2<> cenPos = central.getPos();

(MASA) );
        glBindTexture( GL_TEXTURE_2D, video_.textures.at
        glBegin(GL_QUADS);
        for (unsigned j = 1; j < cant; j++){
            MassProxy mp = sms_.getMassProxy(balls.at(it)->getFirst() + j);
            Vector2<> massPos = mp.getPos();
            MassProxy mp2 = sms_.getMassProxy(balls.at(it)->getFirst() + j + 1);
            Vector2<> massPos2 = mp2.getPos();

            glTexCoord2i( 0, 0 );
            glVertex3f(massPos.x, massPos.y, 0.0f );
            glTexCoord2i( 1, 0 );
            glVertex3f(cenPos.x, cenPos.y, 0.0f );

            glTexCoord2i( 1, 1 );
            glVertex3f(cenPos.x, cenPos.y, 0.0f );

            glTexCoord2i( 0, 1 );
            glVertex3f(massPos2.x, massPos2.y,
0.0f );
        }
        glEnd();
        glBindTexture( GL_TEXTURE_2D, video_.textures.at(9) );
        glBegin(GL_QUADS);
        MassProxy mp = sms_.getMassProxy(balls.at(it)->getLast());
        Vector2<> massPos = mp.getPos();
        MassProxy mp2 = sms_.getMassProxy(balls.at(it)->getFirst() + 1);
        Vector2<> massPos2 = mp2.getPos();

        glTexCoord2i( 0, 0 );
        glVertex3f(massPos.x, massPos.y, 0.0f );
        glTexCoord2i( 1, 0 );
        glVertex3f(cenPos.x, cenPos.y, 0.0f );

        glTexCoord2i( 1, 1 );
        glVertex3f(cenPos.x, cenPos.y, 0.0f );

        glTexCoord2i( 0, 1 );
        glVertex3f(massPos2.x, massPos2.y, 0.0f );

        glEnd();
    }

//IMPRIME COHETES

vector<Element*> cohetes = (sms_.getElementManager())->getCohetes();

int cant;
int longitud;
glBindTexture( GL_TEXTURE_2D, video_.textures.at(COHETE) );
glBegin(GL_QUADS);
if (status==SHOW_TEXT)
    for (it=0; it<cohetes.size();it++){ //para cada una de los cohetes
        cant = cohetes.at(it)->masas.size() - 2;
        longitud = cant / 2;

        for (int i = 0; i < longitud-1; i++)
        {
            MassProxy mp = sms_.getMassProxy(cohetes.at(it)->masas.at(i));
            Vector2<> massPos = mp.getPos();
            MassProxy mp2 = sms_.getMassProxy(cohetes.at(it)->masas.at(i)+1);
            Vector2<> massPos2 = mp2.getPos();
            MassProxy mp3 = sms_.getMassProxy(cohetes.at(it)->masas.at(i)+longitud);
            Vector2<> massPos3 = mp3.getPos();
        }
    }
}

```

```

    MassProxy mp4 = sms_.getMassProxy(cohetes.at(it)->masas.at(i)+longitud+1);
    Vector2<> massPos4 = mp4.getPos();

    glTexCoord2i( 1, 0 );
    glVertex3f( massPos2.x, massPos2.y, 0.0f );
    glTexCoord2i( 1, 1 );
    glVertex3f( massPos.x, massPos.y, 0.0f );
    glTexCoord2i( 0, 1 );
    glVertex3f( massPos3.x, massPos3.y , 0.0f );
    glTexCoord2i( 0, 0 );
    glVertex3f( massPos4.x, massPos4.y, 0.0f );

}
MassProxy mp = sms_.getMassProxy(cohetes.at(it)->masas.at(cohetes.at(it)->masas.size
() -2));
Vector2<> massPos = mp.getPos();
MassProxy mp2 = sms_.getMassProxy(cohetes.at(it)->masas.at(longitud-1));
Vector2<> massPos2 = mp2.getPos();
MassProxy mp3 = sms_.getMassProxy(cohetes.at(it)->masas.at(cohetes.at(it)->masas.size
() -3));
Vector2<> massPos3 = mp3.getPos();

glTexCoord2i( 1, 0 );
glVertex3f( massPos.x, massPos.y, 0.0f );
glTexCoord2i( 1, 1 );
glVertex3f( massPos2.x, massPos2.y, 0.0f );
glTexCoord2i( 0, 1 );
glVertex3f( massPos3.x, massPos3.y , 0.0f );
glTexCoord2i( 0, 0 );
glVertex3f( massPos.x, massPos.y, 0.0f );

}

glEnd();

//IMPRIME RUEDAS
vector<Element*> ruedas = (sms_.getElementManager())->getRuedas();
glBindTexture( GL_TEXTURE_2D, video_.textures.at(RUEDA) );
glBegin(GL_QUADS);
if (status==SHOW_TEXT)
for (it=0; it<ruedas.size();it++){
    MassProxy central = sms_.getMassProxy(ruedas.at(it)->getFirst());
    Vector2<> cenPos = central.getPos();

    for (unsigned j = 1; j < ruedas.at(it)->masas.size()-1; j++){
        MassProxy mp = sms_.getMassProxy(ruedas.at(it)->masas.at(j));
        Vector2<> massPos = mp.getPos();
        MassProxy mp2 = sms_.getMassProxy(ruedas.at(it)->masas.at(j+1));
        Vector2<> massPos2 = mp2.getPos();

        glTexCoord2i( 0, 0 );
        glVertex3f(massPos.x, massPos.y, 0.0f );
        glTexCoord2i( 0, 1 );
        glVertex3f(cenPos.x, cenPos.y, 0.0f );

        glTexCoord2i( 1, 1 );
        glVertex3f(cenPos.x, cenPos.y, 0.0f );

        glTexCoord2i( 1, 0 );
        glVertex3f(massPos2.x, massPos2.y,
0.0f );
    }
}

MassProxy mp = sms_.getMassProxy(ruedas.at(it)->masas.at(ruedas.at(it)-
>masas.size()-1));
Vector2<> massPos = mp.getPos();
MassProxy mp2 = sms_.getMassProxy(ruedas.at(it)->masas.at(1));
Vector2<> massPos2 = mp2.getPos();

glTexCoord2i( 0, 0 );

```

```

        glVertex3f(massPos.x, massPos.y, 0.0f );
        glTexCoord2i( 0, 1 );
        glVertex3f(cenPos.x, cenPos.y, 0.0f );

        glTexCoord2i( 1, 1 );
        glVertex3f(cenPos.x, cenPos.y, 0.0f );

        glTexCoord2i( 1, 0 );
        glVertex3f(massPos2.x, massPos2.y, 0.0f );
    }
    glEnd();

//DIBUJO PUNTOS FIJOS
glBindTexture( GL_TEXTURE_2D, video_.textures.at(PUNTO_FIJO) );
if (status==SHOW_TEXT)
for (size_t i = 0; i < sms_.getNumMasses(); i++)
{
    MassProxy mp = sms_.getMassProxy(i);
    Vector2<> massPos = mp.getPos();
    double invMass = mp.getInvMass();
    double massSize;
    massSize = MASS_SIZE/2;

    glBegin(GL_TRIANGLE_FAN);
    if (invMass==0)

        for (int j = 0; j <= POLY_SIDE; j++){
            glTexCoord2i( 0, 0 );
            glVertex3f(massPos.x, massPos.y, 0.0f );
            glTexCoord2i( 1, 0 );
            glVertex3f(massPos.x, massPos.y, 0.0f );

            glTexCoord2i( 1, 1 );
            glVertex3f(
                massPos.x + massSize *1.4* cos(j * (2 * PI / POLY_SIDE)),
                massPos.y + massSize *1.4* sin(j * (2 * PI / POLY_SIDE)), 0.0f );

            j++;
            glTexCoord2i( 0, 1 );
            glVertex3f(
                massPos.x + massSize *1.4* cos(j * (2 * PI / POLY_SIDE)),
                massPos.y + massSize *1.4* sin(j * (2 * PI / POLY_SIDE)), 0.0f );
        }
        j--;
    }
    glEnd();
}

//INDICA TRANSPARENCIAS PARA AREA DE LLEGADA
glColor4d(1.0, 1.0, 4.0, 0.4);
 glEnable (GL_BLEND);
 glBlendFunc (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

//DIBUJO AREAS DE LLEGADA

vector<ZonaLlegada*> llegadas = (sms_.getElementManager())->getZonasLlegada();
ZonaLlegada* zona;
glBindTexture( GL_TEXTURE_2D, video_.textures.at(8) );
glBegin(GL_QUADS);

for (size_t i= 0 ; i<llegadas.size(); i++)
{
zona = llegadas.at(i);
glTexCoord2i( 0, 0 );
glVertex3f( zona->getX(), zona->getY(), 0.0f );
glTexCoord2i( 1, 0 );
glVertex3f( zona->getX(), zona->getY()-zona->getHeighth(), 0.0f );
glTexCoord2i( 1, 1 );
glVertex3f( zona->getX() + zona->getWidth(), zona->getY()-zona->getHeighth(), 0.0f );
glTexCoord2i( 0, 1 );

```

```
    glVertex3f( zona->getX() + zona->getWidth(), zona->getY(), 0.0f );
}

glEnd();

SDL_GL_SwapBuffers();
glColor4d(1.0, 1.0, 1.0, 1.0);
}
```

```
#include "Soga.h"

Soga::Soga() :
    ElementoLongitudinal("Longitudinal:Soga", Coordenada(long(10), long(10)), 25, 0, 10) {}

Soga::Soga(Cordenada posicion, ulong magnitud, long angulo, long ancho) :
    ElementoLongitudinal("Longitudinal:Soga", posicion, magnitud, angulo, ancho) {}

Soga::~Soga()
{}
```

```
#include "spring_mass_system.h"
#include <cassert>

namespace
{
    double calcDist(const Vector2<>& v1, const Vector2<>& v2)
    {
        Vector2<> d = v2;
        d -= v1;
        return d.norm();
    }

    void inelasticallyCollide(std::vector<Vector2<>>& y, SMSStaticState& x,
                               int m1, int m2, double distance)
    {
        // gets projections of velocity
        Vector2<> axis = y[m2];
        axis -= y[m1];
        double dist = axis.norm();
        axis.normalize();
        double v1p = axis.dot(y[m1 + y.size() / 2]);
        double v2p = axis.dot(y[m2 + y.size() / 2]);

        // checks if the masses really collide
        if (v1p < v2p)
            return;

        // m1 * vf + m2 * vf = m1 * v01 + m2 * v02
        // (1 / m2) * vf + (1 / m1) * vf = (1 / m2) * v01 + (1 / m1) * v02
        // vf = ((1 / m2) * v01 + (1 / m1) * v02) / ((1 / m1) + (1 / m2))
        double vf = (v1p * x.invMassVec[m2] + v2p * x.invMassVec[m1]) /
                    (x.invMassVec[m1] + x.invMassVec[m2]);

        // updates velocities
        Vector2<> dv1 = axis;
        Vector2<> dv2 = axis;
        dv1 *= (vf - v1p);
        dv2 *= (vf - v2p);
        y[m1 + y.size() / 2] += dv1;
        y[m2 + y.size() / 2] += dv2;

        // restores distance
        if (dist < distance)
        {
            Vector2<> dx = axis;
            dx *= (distance - dist) / 2.0;
            y[m1] -= dx;
            y[m2] += dx;
        }
    }

    void handleCollisions(std::vector<Vector2<>>& y, SMSStaticState& x)
    {
        // handles self collisions
        for (size_t i = 0; i < x.collMassesIndices.size(); i++)
        {
            for (size_t j = 0; j < i; j++)
            {
                int m1 = x.collMassesIndices[i];
                int m2 = x.collMassesIndices[j];
                double r1 = x.collMassesRadii[i];
                double r2 = x.collMassesRadii[j];

                if (calcDist(y[m1], y[m2]) >= r1 + r2)
                    continue;

                inelasticallyCollide(y, x, m1, m2, r1 + r2);
            }
        }
    }
}
```

```

void calcYPrime(std::vector<Vector2>& yPrimeVec,
    const std::vector<Vector2>& y,
    SMSStaticState& x, std::vector<Element*> cohetes, std::vector<Element*> ruedas)
{
    // spring forces
    for (size_t i = 0; i < x.springEndPointsVec.size(); i++)
    {
        Vector2 delta(y[x.springEndPointsVec[i].second]);
        delta -= y[x.springEndPointsVec[i].first];

        double forceMag = - x.springKsVec[i] *
            (delta.norm() - x.springNormalLengthsVec[i]);

        // quick hack...
        if (fabs(forceMag) * x.springInvStrength[i] > 1.0)
            x.springKsVec[i] = 0.0;

        delta.normalize();
        delta *= forceMag;
        yPrimeVec[x.springEndPointsVec[i].second + y.size() / 2] += delta;

        delta *= -1.0;
        yPrimeVec[x.springEndPointsVec[i].first + y.size() / 2] += delta;
    }

    // viscous drag
    for (size_t i = 0; i < x.invMassVec.size(); i++)
    {
        Vector2 drag(y[i + y.size() / 2]);
        drag *= x.damping;
        yPrimeVec[i + y.size() / 2] -= drag;
    }

    // dx/dt = v
    for (size_t i = 0; i < yPrimeVec.size() / 2; i++)
        yPrimeVec[i] = y[i + y.size() / 2];

    // dv/dt = force * (1 / mass)
    for (size_t i = 0; i < x.invMassVec.size(); i++)
        yPrimeVec[i + yPrimeVec.size() / 2] *= x.invMassVec[i];

    // adds gravity
    for (size_t i = 0; i < x.invMassVec.size(); i++)
        if (x.invMassVec[i] != 0.0)
            yPrimeVec[i + yPrimeVec.size() / 2] += x.g;

    unsigned it;
    double force, hip, fx, fy, dx, dy;
    force = 1.5;
    for (it=0; it<cohetes.size();it++){ //para cada una de los cohetes
        Vector2 m1 = y[cohetes.at(it)->masas.at(cohetes.at(it)->masas.size()-1)];//mp.getPos();
        Vector2 m2 = y[cohetes.at(it)->masas.at(cohetes.at(it)->masas.size()-2)];//mp.getPos();
        dx = m2.x - m1.x;
        dy = m2.y - m1.y;
        hip = sqrt(pow(dx,2) + pow(dy,2));
        fx = force * (dx/hip);
        fy = force * (dy/hip);

        for (unsigned i = 0; i < cohetes.at(it)->masas.size();i+
+)
            yPrimeVec[cohetes.at(it)->masas.at(i) + yPrimeVec.size() / 2] +=
Vector2 (fx,fy); //x.g;
    }

    force = 1.5;
    double alfa;
    for (it=0; it<ruedas.size();it++){ //para cada una de los cohetes
        Vector2 central = y[ruedas.at(it)->getFirst()];

```

```
        Vector2<> m2;
        double thisforce;
        for (unsigned i = ruedas.at(it)->getFirst() + 1; i <= ruedas.at(it)->getLast()
();i++){
            m2 = y[i];
            dx = m2.x - central.x;
            dy = m2.y - central.y;
            alfa = atan(dy/dx);
            thisforce = force * x.invMassVec[i];
            fx = thisforce * cos (alfa);
            fy = thisforce * sin (alfa);

            if (dy < 0) fy = -fy;
            yPrimeVec[i + yPrimeVec.size() / 2] += Vector2<> (fx,fy);
        }
    }

}

MassProxy::MassProxy(SpringMassSystem& sms, size_t index) :
SMS_(sms), index_(index)
{
}

double MassProxy::getInvMass() const
{
    return SMS_.x_.invMassVec[index_];
}

const Vector2<>& MassProxy::getPos() const
{
    return SMS_.y_[index_];
}

const Vector2<>& MassProxy::getVel() const
{
    return SMS_.y_[index_ + SMS_.y_.size() / 2];
}

void MassProxy::setVel(const Vector2<>& v)
{
    SMS_.y_[index_ + SMS_.y_.size() / 2] = v;
}

SpringProxy::SpringProxy(SpringMassSystem& sms, size_t index) :
SMS_(sms), index_(index)
{
}

const Vector2<>& SpringProxy::getStartPos() const
{
    return SMS_.y_[SMS_.x_.springEndPointsVec[index_].first];
}

const Vector2<>& SpringProxy::getEndPos() const
{
    return SMS_.y_[SMS_.x_.springEndPointsVec[index_].second];
}

double SpringProxy::getRelElongation() const
{
    Vector2<> delta = getEndPos();
    delta -= getStartPos();

    return delta.norm() /
SMS_.x_.springNormalLengthsVec[index_] - 1.0; //estos en 0
}

double SpringProxy::getK() const
```

```
{  
    return SMS_.x_.springKsVec[index_];  
}  
  
SpringMassSystem::SpringMassSystem(const Config& config)  
{  
    x_.g = config.read<Vector2>("SMS.G");  
    x_.damping = config.read<double>("SMS.Damping");  
}  
  
int SpringMassSystem::addMass(double invMass, const Vector2& pos)  
{  
    x_.invMassVec.push_back(invMass);  
    y_.resize(y_.size() + 2);  
    y_[y_.size() / 2 - 1] = pos;  
  
    return x_.invMassVec.size() - 1;  
}  
  
int SpringMassSystem::addCollMass(double invMass, const Vector2& pos,  
                                    double radius)  
{  
    int i = addMass(invMass, pos);  
    x_.collMassesIndices.push_back(i);  
    x_.collMassesRadii.push_back(radius);  
    return i;  
}  
  
void SpringMassSystem::fixMass(int pos)  
{  
    x_.invMassVec.at(pos)=0;  
}  
  
int SpringMassSystem::addSpring(int startMass, int endMass, double k,  
                                 double relElongation, double  
invStrength)  
{  
    x_.springEndPointsVec.push_back(  
        std::make_pair(startMass, endMass));  
    x_.springKsVec.push_back(k);  
  
    Vector2 delta(y_[endMass]);  
    delta -= y_[startMass];  
  
    x_.springNormalLengthsVec.push_back(  
        delta.norm() / (1.0 + relElongation));  
  
    x_.springInvStrength.push_back(invStrength);  
  
    return x_.springKsVec.size() - 1;  
}  
  
size_t SpringMassSystem::getNumMasses() const  
{  
    return x_.invMassVec.size();  
}  
  
size_t SpringMassSystem::getNumSprings() const  
{  
    return x_.springKsVec.size();  
}  
  
MassProxy SpringMassSystem::getMassProxy(size_t index)  
{  
    return MassProxy(*this, index);  
}  
  
SpringProxy SpringMassSystem::getSpringProxy(size_t index)  
{  
    return SpringProxy(*this, index);  
}
```

```
void SpringMassSystem::setMapa(Mapa* mapa){
    this->mapa = mapa;
}

Mapa* SpringMassSystem::getMapa(){
    return this->mapa;
}

bool SpringMassSystem::sameElement(unsigned first,unsigned second){
    unsigned ini,fin;
    for (unsigned i=0; i<manager.vectoresElementos.size();i++)
        for (unsigned j=0; j<manager.vectoresElementos.at(i)->size();j++)
    {
        ini =(manager.vectoresElementos.at(i)->at(j))->getFirst();
        fin =(manager.vectoresElementos.at(i)->at(j))->getLast();
        if ( (first>=ini && first<=fin) && (second>=ini && second<=fin) )
            return true;
    }
    return false;
}

void SpringMassSystem::unifyMass(int first,int second){
    for (unsigned i=0; i<manager.vectoresElementos.size();i++) //en los elementos reemplaza las masas unificadas
        for (unsigned j=0; j<manager.vectoresElementos.at(i)->size();j++)
            for (unsigned k=0; k<(manager.vectoresElementos.at(i)->at(j))->masas.size();k++)
{
                if ( ((manager.vectoresElementos.at(i)->at(j))->masas.at(k) ) ==
first ){
                    ((manager.vectoresElementos.at(i)->at(j))->masas.at(k) ) =
second;
                }
}
    for (unsigned i=0; i<x_.springEndPointsVec.size();i++){
        if (x_.springEndPointsVec.at(i).first == first) {
            x_.springEndPointsVec.at(i).first = second;
        }
        if (x_.springEndPointsVec.at(i).second == first) {
            x_.springEndPointsVec.at(i).second = second;
        }
    }
}

std::vector<int> SpringMassSystem::getVecEnganches(){
    return enganches;
}

void SpringMassSystem::addEnganche(int enganche){
    enganches.push_back(enganche);
}

bool estanCerca(Vector2<> pos1,Vector2<> pos2){

    double modulo,dx,dy;
    dx = pos1.x - pos2.x;
    dy = pos2.y - pos1.y;

    modulo = sqrt(pow(dx,2)+pow(dy,2));
    return (modulo <= 0.04);

}

void SpringMassSystem::checkRoutine(){
    std::vector<int> enganches = getVecEnganches();

    for (size_t i = 0; i< enganches.size() - 1; i++){
        for (size_t j=i+1; j<enganches.size(); j++){
            MassProxy mp2 = getMassProxy(enganches.at(j));
            Vector2<> pos2 = mp2.getPos();
            MassProxy mp = getMassProxy(enganches.at(i));
            Vector2<> pos1 = mp.getPos();
        }
    }
}
```

```

        if (estanCerca(pos1,pos2)){
            if (mp.getInvMass() != 0){
                unifyMass(enganches.at(i),enganches.at(j));           //vuelan i
                y_.at(enganches.at(j)).x=y_.at(enganches.at
(i)).x;
                y_.at(enganches.at(j)).y=y_.at(enganches.at(i)).y;
                y_.at(enganches.at(i)).x=rand()*12;
                y_.at(enganches.at(i)).y=5;
            }
            else {
                unifyMass(enganches.at(j),enganches.at(i));
                y_.at(enganches.at(i)).x=y_.at(enganches.at
(j)).x;
                y_.at(enganches.at(i)).y=y_.at(enganches.at(j)).y;
                y_.at(enganches.at(j)).x=rand()*12;
                y_.at(enganches.at
(j)).y=5;
            }
        }
    }

int mas1, mas2;

for (size_t i = 0; i< x_.springNormalLengthsVec.size();i++){
    mas1 = x_.springEndPointsVec.at(i).first;
    mas2 = x_.springEndPointsVec.at(i).second;

    Vector2<> delta(y_[mas2]);
    delta -= y_[mas1];

    x_.springNormalLengthsVec[i] = (delta.norm());
}
}

double SpringMassSystem::getPotentialEnergy() const
{
    double pe = 0.0;

    Vector2<> deltaVec;
    double deltaNorm;

    // accumulates springs' potential energy
    for (size_t i = 0; i < x_.springEndPointsVec.size(); i++)
    {
        deltaVec = y_[x_.springEndPointsVec[i].second];
        deltaVec -= y_[x_.springEndPointsVec[i].first];
        deltaNorm = deltaVec.norm() - x_.springNormalLengthsVec[i];
        pe += 0.5 * x_.springKsVec[i] * deltaNorm * deltaNorm;
    }

    // accumulates masses' potential energy
    for (size_t i = 0; i < x_.invMassVec.size(); i++)
        if (x_.invMassVec[i] != 0.0)
            pe += -y_[i].dot(x_.g) / x_.invMassVec[i];
}

return pe;
}

double SpringMassSystem::getKineticEnergy() const
{
    double ke = 0.0;

    // accumulates masses' kinetic energy
    for (size_t i = 0; i < y_.size() / 2; i++)
        if (x_.invMassVec[i] != 0.0)
            ke += (0.5 / x_.invMassVec[i]) * y_[i + y_.size() / 2].sqNorm();
        else
            continue;

    return ke;
}

```

```
double SpringMassSystem::getEnergy() const
{
    return getPotentialEnergy() + getKineticEnergy();
}

void SpringMassSystem::eulerStep(double dt)
{
    std::vector<Vector2> > yPrimeVec(y_.size());
    // calculates derivative of system state (y prime)
    calcYPrime(yPrimeVec, y_, x_, manager.getCohetes(), manager.getRuedas());

    // update positions and velocities
    for (size_t i = 0; i < y_.size(); i++)
    {
        Vector2 delta = yPrimeVec[i];
        delta *= dt;
        y_[i] += delta;
    }

    // handles collisions
    handleCollisions(y_, x_);
}

void SpringMassSystem::rk4Step(double dt)
{
    std::vector<Vector2> > k1(y_.size());
    std::vector<Vector2> > k2(y_.size());
    std::vector<Vector2> > k3(y_.size());
    std::vector<Vector2> > k4(y_.size());

    std::vector<Vector2> > yTmp(y_.size());

    // gets k1
    calcYPrime(k1, y_, x_, manager.getCohetes(), manager.getRuedas());

    // gets argument to calculate k2
    for (size_t i = 0; i < yTmp.size(); i++)
    {
        Vector2 delta = k1[i];
        delta *= dt / 2.0;
        yTmp[i] = y_[i];
        yTmp[i] += delta;
    }

    // gets k2
    calcYPrime(k2, yTmp, x_, manager.getCohetes(), manager.getRuedas());

    // gets argument to calculate k3
    for (size_t i = 0; i < yTmp.size(); i++)
    {
        Vector2 delta = k2[i];
        delta *= dt / 2.0;
        yTmp[i] = y_[i];
        yTmp[i] += delta;
    }

    // gets k3
    calcYPrime(k3, yTmp, x_, manager.getCohetes(), manager.getRuedas());

    // gets argument to calculate k4
    for (size_t i = 0; i < yTmp.size(); i++)
    {
        Vector2 delta = k3[i];
        delta *= dt;
        yTmp[i] = y_[i];
        yTmp[i] += delta;
    }

    // gets k4
    calcYPrime(k4, yTmp, x_, manager.getCohetes(), manager.getRuedas());
```

```
// updates state
for (size_t i = 0; i < y_.size(); i++)
{
    Vector2<> delta = k2[i];
    delta += k3[i];
    delta *= 2.0;
    delta += k1[i];
    delta += k4[i];

    delta *= dt / 6.0;

    y_[i] += delta;
}

// handles collisions
handleCollisions(y_, x_);

void SpringMassSystem::debugPrint(std::ostream& os) const
{
    for (size_t i = 0; i < x_.invMassVec.size(); i++)
    {
        os << "Mass " << i << "\n";
        os << "\tInvMass: " << x_.invMassVec[i] << "\n";
        os << "\tPos: " << y_[i] << "\n";
        os << "\tVel: " << y_[i + y_.size() / 2] << "\n";
    }

    for (size_t i = 0; i < x_.springEndPointsVec.size(); i++)
    {
        os << "Spring " << i << "\n";
        os << "\tStart point: " << x_.springEndPointsVec[i].first << "\n";
        os << "\tEnd point: " << x_.springEndPointsVec[i].second << "\n";
        os << "\tK: " << x_.springKsVec[i] << "\n";
        os << "\tNormalLength: " << x_.springNormalLengthsVec[i] << "\n";
    }
}

ElementManager* SpringMassSystem::getElementManager(){
    return &manager;
}
```

```
#include "texture_store.h"
#include <SDL.h>
#include <SDL_image.h>

TextureStore::TTexStore TextureStore::texStore_;

namespace
{
    static void do_loadtexture(const char * aFilename, int clamp = 1)
    {
        int i, j;

        // Load texture using SDL_Image
        SDL_Surface *temp = IMG_Load(aFilename);

        if (temp == NULL)
            return;

        // Set up opengl-compatible pixel format
        SDL_PixelFormat pf;
        pf.palette = NULL;
        pf.BitsPerPixel = 32;
        pf.BytesPerPixel = 4;
        pf.alpha = 0;
        pf.colorkey = 0;
#if SDL_BYTEORDER == SDL_LIL_ENDIAN
        pf.Rmask = 0x000000ff;
        pf.Rshift = 0;
        pf.Rloss = 0;
        pf.Gmask = 0x0000ff00;
        pf.Gshift = 8;
        pf.Gloss = 0;
        pf.Bmask = 0x00ff0000;
        pf.Bshift = 16;
        pf.Bloss = 0;
        pf.Amask = 0xff000000;
        pf.Ashift = 24;
        pf.Aloss = 0;
#else
        pf.Amask = 0x000000ff;
        pf.Ashift = 0;
        pf.Aloss = 0;
        pf.Bmask = 0x0000ff00;
        pf.Bshift = 8;
        pf.Bloss = 0;
        pf.Gmask = 0x00ff0000;
        pf.Gshift = 16;
        pf.Gloss = 0;
        pf.Rmask = 0xff000000;
        pf.Rshift = 24;
        pf.Rloss = 0;
#endif
        // Convert texture to said format
        SDL_Surface *tm = SDL_ConvertSurface(temp, &pf, SDL_SWSURFACE);

        // Cleanup
        SDL_FreeSurface(temp);

        // Lock the converted surface for reading
        SDL_LockSurface(tm);

        int w,h,l;
        w = tm->w;
        h = tm->h;
        l = 0;
        unsigned int * mip = new unsigned int[w * h * 5];
        unsigned int * src = (unsigned int *)tm->pixels;

        memset(mip, 0, w*h*4);

        // mark all pixels with alpha = 0 to black
        for (i = 0; i < h; i++)
            for (j = 0; j < w; j++)
                if ((src[i*w+j] & 0x000000ff) == 0)
                    mip[i*w+j] = 0;
    }
}
```

```

    {
        for (j = 0; j < w; j++)
        {
            if ((src[i*w+j] & pf.Amask) == 0)
                src[i*w+j] = 0;
        }
    }

    // Tell OpenGL to read the texture
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, tm->w, tm->h, 0, GL_RGBA, GL_UNSIGNED_BYTE,
(GLvoid*)src);

if (mip)
{
    // precalculate summed area tables
    // it's a box filter, which isn't very good, but at least it's fast =)
    int ra = 0, ga = 0, ba = 0, aa = 0;
    int i, j, c;
    unsigned int * rbuf = mip + (tm->w * tm->h * 1);
    unsigned int * gbuf = mip + (tm->w * tm->h * 2);
    unsigned int * bbuf = mip + (tm->w * tm->h * 3);
    unsigned int * abuf = mip + (tm->w * tm->h * 4);

    for (j = 0, c = 0; j < tm->h; j++)
    {
        ra = ga = ba = aa = 0;
        for (i = 0; i < tm->w; i++, c++)
        {
            ra += (src[c] >> 0) & 0xff;
            ga += (src[c] >> 8) & 0xff;
            ba += (src[c] >> 16) & 0xff;
            aa += (src[c] >> 24) & 0xff;
            if (j == 0)
            {
                rbuf[c] = ra;
                gbuf[c] = ga;
                bbuf[c] = ba;
                abuf[c] = aa;
            }
            else
            {
                rbuf[c] = ra + rbuf[c - tm->w];
                gbuf[c] = ga + gbuf[c - tm->w];
                bbuf[c] = ba + bbuf[c - tm->w];
                abuf[c] = aa + abuf[c - tm->w];
            }
        }
    }
}

while (w > 1 || h > 1)
{
    l++;
    w /= 2;
    h /= 2;
    if (w == 0) w = 1;
    if (h == 0) h = 1;

    int dw = tm->w / w;
    int dh = tm->h / h;

    for (j = 0, c = 0; j < h; j++)
    {
        for (i = 0; i < w; i++, c++)
        {
            int x1 = i * dw;
            int y1 = j * dh;
            int x2 = x1 + dw - 1;
            int y2 = y1 + dh - 1;
            int div = (x2 - x1) * (y2 - y1);
            y1 *= tm->w;
            y2 *= tm->w;
        }
    }
}

```

```

+ rbuf[y1 + x1];
+ gbuf[y1 + x1];
+ bbuf[y1 + x1];
+ abuf[y1 + x1];

        int r = rbuf[y2 + x2] - rbuf[y1 + x2] - rbuf[y2 + x1];
        int g = gbuf[y2 + x2] - gbuf[y1 + x2] - gbuf[y2 + x1];
        int b = bbuf[y2 + x2] - bbuf[y1 + x2] - bbuf[y2 + x1];
        int a = abuf[y2 + x2] - abuf[y1 + x2] - abuf[y2 + x1];

        r /= div;
        g /= div;
        b /= div;
        a /= div;

        if (a == 0)
            mip[c] = 0;
        else
            mip[c] = ((r & 0xff) << 0) |
                ((g & 0xff) << 8) |
                ((b & 0xff) << 16) |
                ((a & 0xff) << 24));
    }
    glTexImage2D(GL_TEXTURE_2D, l, GL_RGBA, w, h, 0, GL_RGBA,
GL_UNSIGNED_BYTE, (GLvoid*)mip);
}
glTexParameteri
(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR_MIPMAP_LINEAR); // Linear Filtering
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LINEAR); // Linear Filtering
delete[] mip;
}
else
{
    glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR); // Linear Filtering
    glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LINEAR); // Linear Filtering
}

// Unlock..
SDL_UnlockSurface(tm);

// and cleanup.
SDL_FreeSurface(tm);

if (clamp)
{
    // Set up texture parameters
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);
}
else
{
    // Set up texture parameters
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
}

GLuint load_texture(const char* aFilename, int clamp = 1)
{
    // Create OpenGL texture handle and bind it to use

    GLuint texname;
    glGenTextures(1,&texname);
    glBindTexture(GL_TEXTURE_2D,texname);

    do_loadtexture(aFilename);

    return texname;
}

```

```
    }

GLuint TextureStore::addTexture(const std::string& filename)
{
    TTexStore::iterator it = texStore_.find(filename);

    // checks if it's already stored
    if (it != texStore_.end())
        return it->second;

    // creates the texture
    GLuint texID = load_texture(filename.c_str());

    // stores the id
    texStore_[filename] = texID;

    // returns the ID
    return texID;
}
```

```
#include "Ventana.h"
#include <iostream>
#include "Boton.h"
#include <vector>
#include "InterfazMapa.h"
#include "ParserXML.h"
#include "sms_app.h"
using namespace std;

Ventana::Ventana(int ancho, int alto) {

    this->alto = alto;
    this->ancho = ancho;
    this->salir = false;
    this->sms = false;

    int w = ancho;
    int h = alto;
    int depth = 16;

    SDL_InitSubSystem(SDL_INIT_VIDEO);
    SDL_GL_SetAttribute(SDL_GL_DOUBLEBUFFER, 1);
    pantalla = SDL_SetVideoMode(w, h, depth, SDL_OPENGL | SDL_FULLSCREEN);
    SDL_WM_SetCaption("Editor de Escenarios", NULL);

    double ar = static_cast<double>(w) / h;
    glOrtho(-ar, ar, -1.0, 1.0, -1.0, 1.0);
    glEnable(GL_TEXTURE_2D);

    glClearColor( 0.0f, 0.0f, 0.0f, 0.0f);
    glClear(GL_COLOR_BUFFER_BIT);

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();

    glClearColor(1.0f, 1.0f, 1.0f, 0.0f);

}

Ventana::~Ventana() {

    list<PanelContenedor*>::iterator it;
    for (it = _paneles.begin(); it != _paneles.end(); it++) {
        delete (*it);
    }

    SDL_QuitSubSystem(SDL_INIT_VIDEO);
    SDL_Quit();
}

void Ventana::ejecutar() {

    SDL_Event eventos;
    Coordenada posicionMouse;

    while (!salir) {
        imprimir_pantalla();

        if (SDL_WaitEvent(&eventos)) {
            posicionMouse.setX(eventos.motion.x);
            posicionMouse.setY(eventos.motion.y);
            switch (eventos.type) {
                case SDL_QUIT:
                    salir = true;
                    break;

                case SDL_MOUSEBUTTONDOWN:
                    seEjecutaEventoEn(posicionMouse, SDL_MOUSEBUTTONDOWN);
            }
        }
    }
}
```

```
        break;

    case SDL_MOUSEBUTTONDOWN:
        seEjecutaEventoEn(posicionMouse, SDL_MOUSEBUTTONDOWN);
        break;

    case SDL_MOUSEMOTION:
        seEjecutaEventoEn(posicionMouse, SDL_MOUSEMOTION);
        break;

    default:
        break;
    }
} else {
    cout << "Error " << SDL_GetError() << endl;
    salir = true;
}
}

void Ventana::agregarPanel(PanelContenedor * panel, const Coordenada posicion) {
    panel->setVentana(this);
    panel->setPantalla(pantalla);
    panel->setPosicion(posicion);
    panel->setId(_paneles.size() + 1);
    _paneles.push_back(panel);
}

void Ventana::agregarPanel(PanelContenedor * panel, const int lugar) {
    Coordenada pos;
    panel->setVentana(this);
    switch (lugar) {
    case 0:
        pos.setX(20 + _paneles.size() * (panel->getAncho() + 20));
        pos.setY(this->alto - panel->getAltura());
        break;
    case 1:
        pos.setX(0);
        pos.setY(this->alto - panel->getAltura());
    }
    panel->setPantalla(pantalla);
    panel->setPosicion(pos);
    panel->setId(_paneles.size() + 1);
    _paneles.push_back(panel);
}

void Ventana::imprimir_pantalla(void) {

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    list<PanelContenedor*>::iterator it;
    for (it = _paneles.begin(); it != _paneles.end(); it++) {
        (*it)->imprimir();
    }

    SDL_GL_SwapBuffers();
}

bool Ventana::seEjecutaEventoEn(const Coordenada punto, int evento) {
    list<PanelContenedor*>::iterator it;
    for (it = _paneles.begin(); it != _paneles.end(); it++) {
        if ((*it)->pertenece(punto)) {
            if ((*it)->seEjecutaEventoEn(punto, evento)) {
                return false;
            }
        }
    }
}
```

```
        }

    }

    switch(evento) {
        case SDL_MOUSEBUTTONDOWN: evento_MouseDown(punto); break;
        case SDL_MOUSEMOTION: evento_MouseOver(punto); break;
        case SDL_MOUSEBUTTONUP: evento_MouseUp(punto); break;
    }

    return true;
}

void Ventana::evento_MouseDown(Cordenada punto) {

}

void Ventana::evento_MouseOver(Cordenada punto) {

    list<PanelContenedor*>::iterator it;
    for (it = _paneles.begin(); it != _paneles.end(); it++)
        if((*it)->getNombre().find("Boton") != string::npos) {
            Boton * boton = (Boton*) (*it);
            boton->setMouseOver(false);
            if(boton->esta_presionado())
                boton->setImagenDeFondo(TEXTURA_MOUSEDOWN);
            else
                boton->setImagenDeFondo(TEXTURA_DEFECTO);
        }
}

void Ventana::evento_MouseUp(Cordenada punto) {

}

void Ventana::desclickearBotones(int id_excepcion) {
    list<PanelContenedor*>::iterator it;
    for (it = _paneles.begin(); it != _paneles.end(); it++)
        if((*it)->getNombre().find("Boton") != string::npos) {
            Boton * boton = (Boton*) (*it);
            if(boton->getId() == id_excepcion) {
                continue;
            } else {
                boton->desapretar();
                boton->setImagenDeFondo(TEXTURA_DEFECTO);
            }
        }
}

void Ventana::corregirCoordenadas(ulong *x, ulong *y) {
    *x = (*x - (ancho/2.0)) / 300;
    *y = (-*y + (alto/2.0)) / 300;
}
```

```
#include "VentanaConfiguracion.h"
#include "ParserXML.h"
#include <glib.h>
#include <glib/gprintf.h>
#include <gtk/gtk.h>
#include <string>
#include <ctype.h>

using namespace std;

bool VentanaConfiguracion::textoVacio(const string& texto)
{
    unsigned int i;
    bool retorno = true;
    for (i = 0 ; i < texto.length() ; i++)
    {
        if (texto[i] != ' ')
        {
            retorno = false;
            break;
        }
    }
    return retorno;
}

bool VentanaConfiguracion::esNumero(const std::string& cadena)
{
    //analiza letra por letra si es no cadena
    for (unsigned int i = 0; i < cadena.length(); i++)
    {
        if (!isdigit(cadena[i]))
            return false;
    }
    return true;
}

void VentanaConfiguracion::mostrarAlerta(string& leyenda, GtkWidget* ventanaPadre)
{
    GtkWidget *alerta = gtk_dialog_new ();
    GtkWidget* label = gtk_label_new (leyenda.c_str());

    gtk_box_pack_start (GTK_BOX (GTK_DIALOG (alerta)->vbox), label, FALSE, FALSE, 0);

    GtkWidget* boton = gtk_button_new_from_stock("Aceptar");
    gtk_box_pack_start (GTK_BOX (GTK_DIALOG (alerta)->action_area), boton, FALSE, FALSE, 0);

    gtk_window_set_title(GTK_WINDOW(alerta),"TatuCarreta");

    gtk_window_set_transient_for (GTK_WINDOW(alerta),GTK_WINDOW(ventanaPadre));
    gtk_window_set_position(GTK_WINDOW(alerta),GTK_WIN_POS_CENTER_ON_PARENT);

    g_signal_connect(G_OBJECT(boton), "clicked", G_CALLBACK(cerrar_ventana), alerta);
}

void VentanaConfiguracion::guardar(GtkWidget *widget, gpointer data)
{
    datosConfiguracion* datos = (datosConfiguracion*) data;

    ParserXML parser;

    // obtiene todas las entradas de datos
    string nombre = gtk_entry_get_text(GTK_ENTRY(datos->textoNom));
    string plata = gtk_entry_get_text(GTK_ENTRY(datos->textoPlata));
    string plataMasa = gtk_entry_get_text(GTK_ENTRY(datos->textoPlataMasa));
    string plataPuntoFijo = gtk_entry_get_text(GTK_ENTRY(datos->textoPlataPuntoFijo));
    string plataRueda = gtk_entry_get_text(GTK_ENTRY(datos->textoPlataRueda));
    string plataBarra = gtk_entry_get_text(GTK_ENTRY(datos->textoPlataBarra));
    string plataPlataforma = gtk_entry_get_text(GTK_ENTRY(datos->textoPlataPlataforma));
```

```
string plataSoga = gtk_entry_get_text(GTK_ENTRY(datos->textoPlataSoga));
string plataCinta = gtk_entry_get_text(GTK_ENTRY(datos->textoPlataCinta));
string plataCohete = gtk_entry_get_text(GTK_ENTRY(datos->textoPlataCohete));

// entradas en entero
int plataEnteroMasa;
int plataEnteroPuntoFijo;
int plataEnteroRueda;
int plataEnteroBarra;
int plataEnteroPlataforma;
int plataEnteroSoga;
int plataEnteroCinta;
int plataEnteroCohete;

// datos del tiempo de cohete
string tiempoCohete = gtk_entry_get_text(GTK_ENTRY(datos->textoTiempoCohete));
int tiempoEnteroCohete;

int plataEntero;
string archivo = "./escenarios/" + nombre + ".xml";

// analiza si le faltó ingresar algún dato (casilla vacía)
if ((textoVacio(nombre)) || (textoVacio(plata)) || (textoVacio(plataMasa)) || (textoVacio(plataPuntoFijo)) || (textoVacio(plataRueda)) || (textoVacio(plataBarra)) || (textoVacio(plataPlataforma)) || (textoVacio(plataSoga)) || (textoVacio(plataCinta)) || (textoVacio(plataCohete)))
{
    string leyenda = "Faltan ingresar datos";
    mostrarAlerta(leyenda,datos->ventana);
    return;
}

// analiza si los datos ingresados en casillas numéricas son efectivamente numéricos
if (!esNumero(plata) || !esNumero(plataMasa) || !esNumero(plataPuntoFijo)
    || !esNumero(plataRueda) || !esNumero(plataBarra) || !esNumero(plataPlataforma)
    || !esNumero(plataSoga) || !esNumero(plataCinta) || !esNumero(plataCohete)
    || !esNumero(tiempoCohete))
{
    string leyenda = "Datos numéricos incorrectos";
    mostrarAlerta(leyenda,datos->ventana);
    return;
}

// obtiene los datos ingresados en forma de entero
plataEntero = atoi(plata.c_str());
plataEnteroMasa = atoi(plataMasa.c_str());
plataEnteroPuntoFijo = atoi(plataPuntoFijo.c_str());
plataEnteroRueda = atoi(plataRueda.c_str());
plataEnteroBarra = atoi(plataBarra.c_str());
plataEnteroPlataforma = atoi(plataPlataforma.c_str());
plataEnteroSoga = atoi(plataSoga.c_str());
plataEnteroCinta = atoi(plataCinta.c_str());
plataEnteroCohete = atoi(plataCohete.c_str());
tiempoEnteroCohete = atoi(tiempoCohete.c_str());

// analiza si fue un ingreso válido en entero
if ((plataEntero < 0) || (plataEnteroMasa < 0) || (plataEnteroPuntoFijo < 0) ||
    (plataEnteroBarra < 0) || (plataEnteroPlataforma < 0) || (plataEnteroSoga < 0) || (plataEnteroCinta < 0) ||
    (plataEnteroCohete < 0))
{
    string leyenda = "Dinero incorrecto";
    mostrarAlerta(leyenda,datos->ventana);
    return;
}

if (tiempoEnteroCohete < 0)
{
    string leyenda = "Tiempo de cohete incorrecto";
    mostrarAlerta(leyenda,datos->ventana);
    return;
}
```

```

//Modifica el mapa
datos->pthis->mapa->setNombre(nombre);
datos->pthis->mapa->setPlata(plataEntero);

//Analiza si se habilitó o no cada elemento
if (!GTK_TOGGLE_BUTTON (datos->elMasa)->active)
    datos->pthis->mapa->deshabilitarElemento(MASA);
else
    datos->pthis->mapa->habilitarElemento(MASA);
if (!GTK_TOGGLE_BUTTON (datos->elPuntoFijo)->active)
    datos->pthis->mapa->deshabilitarElemento(PUNTO_FIJO);
else
    datos->pthis->mapa->habilitarElemento(PUNTO_FIJO);
if (!GTK_TOGGLE_BUTTON (datos->elRueda)->active)
    datos->pthis->mapa->deshabilitarElemento(RUEDA);
else
    datos->pthis->mapa->habilitarElemento(RUEDA);
if (!GTK_TOGGLE_BUTTON (datos->elBarra)->active)
    datos->pthis->mapa->deshabilitarElemento(BARRA);
else
    datos->pthis->mapa->habilitarElemento(BARRA);
if (!GTK_TOGGLE_BUTTON (datos->elPlataforma)->active)
    datos->pthis->mapa->deshabilitarElemento(PLATAFORMA);
else
    datos->pthis->mapa->habilitarElemento(PLATAFORMA);
if (!GTK_TOGGLE_BUTTON (datos->elSoga)->active)
    datos->pthis->mapa->deshabilitarElemento(SOGA);
else
    datos->pthis->mapa->habilitarElemento(SOGA);
if (!GTK_TOGGLE_BUTTON (datos->elCinta)->active)
    datos->pthis->mapa->deshabilitarElemento(CINTA);
else
    datos->pthis->mapa->habilitarElemento(CINTA);
if (!GTK_TOGGLE_BUTTON (datos->elCohete)->active)
    datos->pthis->mapa->deshabilitarElemento(COHETE);
else
    datos->pthis->mapa->habilitarElemento(COHETE);

//Setea el costo de cada elemento
datos->pthis->mapa->setPrecio(MASA,plataEnteroMasa);
datos->pthis->mapa->setPrecio(PUNTO_FIJO,plataEnteroPuntoFijo);
datos->pthis->mapa->setPrecio(RUEDA,plataEnteroRueda);
datos->pthis->mapa->setPrecio(BARRA,plataEnteroBarra);
datos->pthis->mapa->setPrecio(PLATAFORMA,plataEnteroPlataforma);
datos->pthis->mapa->setPrecio(SOGA,plataEnteroSoga);
datos->pthis->mapa->setPrecio(CINTA,plataEnteroCinta);
datos->pthis->mapa->setPrecio(COHETE,plataEnteroCohete);

//Setea el tiempo del cohete
datos->pthis->mapa->setTiempoCohete(tiempoEnteroCohete);

parser.obtenerXMLArchivo(archivo,datos->pthis->mapa);

gtk_widget_destroy(datos->ventana);

gtk_main_quit();
}

```

```

VentanaConfiguracion::VentanaConfiguracion(Mapa* mapa, GtkWidget* ventanaPpal)
{
    this->mapa = mapa;

    // ventana
    GtkWidget* ventana = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_window_set_position(GTK_WINDOW(ventana),GTK_WIN_POS_CENTER);
    gtk_container_set_border_width(GTK_CONTAINER(ventana), 10);
    gtk_window_set_title (GTK_WINDOW (ventana), "Editor :: TatuCarreta Run");
    gtk_window_set_default_size (GTK_WINDOW (ventana), 300, 100);
    gtk_window_set_resizable(GTK_WINDOW (ventana),FALSE);
}

```

```
// etiquetas
GtkWidget* etNombre = gtk_label_new ("Nombre del escenario");
GtkWidget* etPlata = gtk_label_new ("Dinero disponible");
GtkWidget* etTitulo = gtk_label_new ("Elementos disponibles");

GtkWidget* etCostoMasa = gtk_label_new (" | Costo:");
GtkWidget* etCostoPuntoFijo = gtk_label_new (" | Costo:");
GtkWidget* etCostoRueda = gtk_label_new (" | Costo:");
GtkWidget* etCostoBarra = gtk_label_new (" | Costo:");
GtkWidget* etCostoPlataforma = gtk_label_new (" | Costo:");
GtkWidget* etCostoSoga = gtk_label_new (" | Costo:");
GtkWidget* etCostoCinta = gtk_label_new (" | Costo:");
GtkWidget* etCostoCohete = gtk_label_new (" | Costo:");
GtkWidget* etTiempoCohete = gtk_label_new (" | Tiempo en ms (cero para infinito):");

// casillas de entrada de texto
GtkWidget* textoNom = gtk_entry_new();
gtk_entry_set_width_chars(GTK_ENTRY(textoNom), 20);
GtkWidget* textoPlata = gtk_entry_new();
gtk_entry_set_width_chars(GTK_ENTRY(textoPlata), 20);
GtkWidget* textoPlataMasa = gtk_entry_new();
gtk_entry_set_width_chars(GTK_ENTRY(textoPlataMasa), 20);
GtkWidget* textoPlataPuntoFijo = gtk_entry_new();
gtk_entry_set_width_chars(GTK_ENTRY(textoPlataPuntoFijo), 20);
GtkWidget* textoPlataRueda = gtk_entry_new();
gtk_entry_set_width_chars(GTK_ENTRY(textoPlataRueda), 20);
GtkWidget* textoPlataBarra = gtk_entry_new();
gtk_entry_set_width_chars(GTK_ENTRY(textoPlataBarra), 20);
GtkWidget* textoPlataPlataforma = gtk_entry_new();
gtk_entry_set_width_chars(GTK_ENTRY(textoPlataPlataforma), 20);
GtkWidget* textoPlataSoga = gtk_entry_new();
gtk_entry_set_width_chars(GTK_ENTRY(textoPlataSoga), 20);
GtkWidget* textoPlataCinta = gtk_entry_new();
gtk_entry_set_width_chars(GTK_ENTRY(textoPlataCinta), 20);
GtkWidget* textoPlataCohete = gtk_entry_new();
gtk_entry_set_width_chars(GTK_ENTRY(textoPlataCohete), 20);
GtkWidget* textoTiempoCohete = gtk_entry_new();
gtk_entry_set_width_chars(GTK_ENTRY(textoTiempoCohete), 20);

// botones de elección
GtkWidget* elMasa = gtk_check_button_new_with_label ("Masas");
GtkWidget* elPuntoFijo = gtk_check_button_new_with_label ("Puntos fijos");
GtkWidget* elRueda = gtk_check_button_new_with_label ("Ruedas");
GtkWidget* elBarra = gtk_check_button_new_with_label ("Barras metálicas");
GtkWidget* elPlataforma = gtk_check_button_new_with_label ("Plataformas metálicas");
GtkWidget* elSoga = gtk_check_button_new_with_label ("Sogas");
GtkWidget* elCinta = gtk_check_button_new_with_label ("Cintas de lona");
GtkWidget* elCohete = gtk_check_button_new_with_label ("Cohetes");

gtk_entry_set_text(GTK_ENTRY(textoNom), (mapa->getNombre()).c_str());

// inicializa los botones de elección
gtk_toggle_button_set_active(GTK_TOGGLE_BUTTON(elMasa), mapa->elementoHabilitado(MASA));
gtk_toggle_button_set_active(GTK_TOGGLE_BUTTON(elPuntoFijo), mapa->elementoHabilitado(PUNTO_FIJO));
gtk_toggle_button_set_active(GTK_TOGGLE_BUTTON(elRueda), mapa->elementoHabilitado(RUEDA));
gtk_toggle_button_set_active(GTK_TOGGLE_BUTTON(elBarra), mapa->elementoHabilitado(BARRA));
gtk_toggle_button_set_active(GTK_TOGGLE_BUTTON(elPlataforma), mapa->elementoHabilitado(PLATAFORMA));
gtk_toggle_button_set_active(GTK_TOGGLE_BUTTON(elSoga), mapa->elementoHabilitado(SOGA));
gtk_toggle_button_set_active(GTK_TOGGLE_BUTTON(elCinta), mapa->elementoHabilitado(CINTA));
gtk_toggle_button_set_active(GTK_TOGGLE_BUTTON(elCohete), mapa->elementoHabilitado(COHETE));

string numero;

// inicializa las entradas
enteroAString(numero, mapa->getPlata());
gtk_entry_set_text(GTK_ENTRY(textoPlata), numero.c_str());
enteroAString(numero, mapa->getPrecio(MASA));
gtk_entry_set_text(GTK_ENTRY(textoPlataMasa), numero.c_str());
```

```
enteroAString(numero, mapa->getPrecio(PUNTO_FIJO));
gtk_entry_set_text(GTK_ENTRY(textoPlataPuntoFijo), numero.c_str());
enteroAString(numero, mapa->getPrecio(RUEDA));
gtk_entry_set_text(GTK_ENTRY(textoPlataRueda), numero.c_str());
enteroAString(numero, mapa->getPrecio(BARRA));
gtk_entry_set_text(GTK_ENTRY(textoPlataBarra), numero.c_str());
enteroAString(numero, mapa->getPrecio(PLATAFORMA));
gtk_entry_set_text(GTK_ENTRY(textoPlataPlataforma), numero.c_str());
enteroAString(numero, mapa->getPrecio(SOGA));
gtk_entry_set_text(GTK_ENTRY(textoPlataSoga), numero.c_str());
enteroAString(numero, mapa->getPrecio(CINTA));
gtk_entry_set_text(GTK_ENTRY(textoPlataCinta), numero.c_str());
enteroAString(numero, mapa->getPrecio(COHETE));
gtk_entry_set_text(GTK_ENTRY(textoPlataCohete), numero.c_str());
enteroAString(numero, mapa->getTiempoCohete());
gtk_entry_set_text(GTK_ENTRY(textoTiempoCohete), numero.c_str());

// botón guardar
GtkWidget* boton = gtk_button_new_from_stock("Guardar escenario");

GtkWidget* contenedor = gtk_vbox_new(FALSE, 10);
GtkWidget* contenedorNom = gtk_hbox_new(FALSE, 10);
GtkWidget* contenedorPlata = gtk_hbox_new(FALSE, 10);

//Contenedores
GtkWidget* contenedorMasa = gtk_hbox_new(FALSE, 10);
GtkWidget* contenedorPuntoFijo = gtk_hbox_new(FALSE, 10);
GtkWidget* contenedorRueda = gtk_hbox_new(FALSE, 10);
GtkWidget* contenedorBarra = gtk_hbox_new(FALSE, 10);
GtkWidget* contenedorPlataforma = gtk_hbox_new(FALSE, 10);
GtkWidget* contenedorSoga = gtk_hbox_new(FALSE, 10);
GtkWidget* contenedorCinta = gtk_hbox_new(FALSE, 10);
GtkWidget* contenedorCohete = gtk_hbox_new(FALSE, 10);

// empaqueta los elementos
gtk_box_pack_start (GTK_BOX(contenedorNom), etNombre, FALSE, FALSE, 3);
gtk_box_pack_start (GTK_BOX(contenedorNom), textoNom, TRUE, TRUE, 3);

gtk_box_pack_start (GTK_BOX(contenedorPlata), etPlata, FALSE, FALSE, 3);
gtk_box_pack_start (GTK_BOX(contenedorPlata), textoPlata, TRUE, TRUE, 3);

gtk_box_pack_start (GTK_BOX(contenedor), contenedorNom, FALSE, FALSE, 3);
gtk_box_pack_start (GTK_BOX(contenedor), contenedorPlata, FALSE, FALSE, 3);
gtk_box_pack_start (GTK_BOX(contenedor), etTitulo, FALSE, FALSE, 3);
gtk_box_pack_start (GTK_BOX(contenedor), contenedorMasa, FALSE, FALSE, 3);
gtk_box_pack_start (GTK_BOX(contenedor), contenedorPuntoFijo, FALSE, FALSE, 3);
gtk_box_pack_start (GTK_BOX(contenedor), contenedorRueda, FALSE, FALSE, 3);
gtk_box_pack_start (GTK_BOX(contenedor), contenedorBarra, FALSE, FALSE, 3);
gtk_box_pack_start (GTK_BOX(contenedor), contenedorPlataforma, FALSE, FALSE, 3);
gtk_box_pack_start (GTK_BOX(contenedor), contenedorSoga, FALSE, FALSE, 3);
gtk_box_pack_start (GTK_BOX(contenedor), contenedorCinta, FALSE, FALSE, 3);
gtk_box_pack_start (GTK_BOX(contenedor), contenedorCohete, FALSE, FALSE, 3);

gtk_box_pack_start (GTK_BOX(contenedorMasa), elMasa, FALSE, FALSE, 3);
gtk_box_pack_start (GTK_BOX(contenedorMasa), etCostoMasa, FALSE, FALSE, 3);
gtk_box_pack_start (GTK_BOX(contenedorMasa), textoPlataMasa, TRUE, TRUE, 3);

gtk_box_pack_start (GTK_BOX(contenedorPuntoFijo), elPuntoFijo, FALSE, FALSE, 3);
gtk_box_pack_start (GTK_BOX(contenedorPuntoFijo), etCostoPuntoFijo, FALSE, FALSE, 3);
gtk_box_pack_start (GTK_BOX(contenedorPuntoFijo), textoPlataPuntoFijo, TRUE, TRUE, 3);

gtk_box_pack_start (GTK_BOX(contenedorRueda), elRueda, FALSE, FALSE, 3);
gtk_box_pack_start (GTK_BOX(contenedorRueda), etCostoRueda, FALSE, FALSE, 3);
gtk_box_pack_start (GTK_BOX(contenedorRueda), textoPlataRueda, TRUE, TRUE, 3);

gtk_box_pack_start (GTK_BOX(contenedorBarra), elBarra, FALSE, FALSE, 3);
gtk_box_pack_start (GTK_BOX(contenedorBarra), etCostoBarra, FALSE, FALSE, 3);
gtk_box_pack_start (GTK_BOX(contenedorBarra), textoPlataBarra, TRUE, TRUE, 3);

gtk_box_pack_start (GTK_BOX(contenedorPlataforma), elPlataforma, FALSE, FALSE, 3);    401
```

```
gtk_box_pack_start (GTK_BOX(contenedorPlataforma), etCostoPlataforma, FALSE, FALSE, 3);
gtk_box_pack_start (GTK_BOX(contenedorPlataforma), textoPlataPlataforma, TRUE, TRUE, 3);

gtk_box_pack_start (GTK_BOX(contenedorSoga), elSoga, FALSE, FALSE, 3);
gtk_box_pack_start (GTK_BOX(contenedorSoga), etCostoSoga, FALSE, FALSE, 3);
gtk_box_pack_start (GTK_BOX(contenedorSoga), textoPlataSoga, TRUE, TRUE, 3);

gtk_box_pack_start (GTK_BOX(contenedorCinta), elCinta, FALSE, FALSE, 3);
gtk_box_pack_start (GTK_BOX(contenedorCinta), etCostoCinta, FALSE, FALSE, 3);
gtk_box_pack_start (GTK_BOX(contenedorCinta), textoPlataCinta, TRUE, TRUE, 3);

gtk_box_pack_start (GTK_BOX(contenedorCohete), elCohete, FALSE, FALSE, 3);
gtk_box_pack_start (GTK_BOX(contenedorCohete), etTiempoCohete, FALSE, FALSE, 3);
gtk_box_pack_start (GTK_BOX(contenedorCohete), textoTiempoCohete, TRUE, TRUE, 3);
gtk_box_pack_start (GTK_BOX(contenedorCohete), etCostoCohete, FALSE, FALSE, 3);
gtk_box_pack_start (GTK_BOX(contenedorCohete), textoPlataCohete, TRUE, TRUE, 3);

gtk_box_pack_start (GTK_BOX(contenedor), boton, FALSE, FALSE, 3);
gtk_container_add(GTK_CONTAINER(ventana), contenedor);

datos = new datosConfiguracion();
datos->textoNom = textoNom;
datos->textoPlata = textoPlata;
datos->pthis = this;
datos->elMasa = elMasa;
datos->elPuntoFijo = elPuntoFijo;
datos->elRueda = elRueda;
datos->elBarra = elBarra;
datos->elPlataforma = elPlataforma;
datos->elSoga = elSoga;
datos->elCinta = elCinta;
datos->elCohete = elCohete;
datos->ventana = ventana;

datos->textoPlataMasa = textoPlataMasa;
datos->textoPlataPuntoFijo = textoPlataPuntoFijo;
datos->textoPlataRueda = textoPlataRueda;
datos->textoPlataBarra = textoPlataBarra;
datos->textoPlataPlataforma = textoPlataPlataforma;
datos->textoPlataSoga = textoPlataSoga;
datos->textoPlataCinta = textoPlataCinta;
datos->textoPlataCohete = textoPlataCohete;
datos->textoTiempoCohete = textoTiempoCohete;

// conexión de señales
g_signal_connect(G_OBJECT(boton), "clicked", G_CALLBACK(guardar), datos);
g_signal_connect(G_OBJECT(ventana), "destroy", G_CALLBACK(destruir), datos);

gtk_widget_show_all (ventana);

}

VentanaConfiguracion::~VentanaConfiguracion()
{
    delete datos->pthis->mapa;
    delete datos;
}

void VentanaConfiguracion::destruir(GtkWidget *widget, gpointer data)
{
    gtk_main_quit();
}

void VentanaConfiguracion::cerrar_ventana(GtkWidget *widget, gpointer data)
{
    GtkWidget* vent = (GtkWidget*) data;
    gtk_widget_destroy(vent);
}
```

```
void VentanaConfiguracion::enteroAString(std::string& cadena, ulong entero)
{
    if (entero == 0)
        cadena = "0";
    else
        cadena = "";
    while (entero != 0)
    {
        switch (entero %10)
        {
            case 0: cadena="0"+cadena; break;
            case 1: cadena="1"+cadena; break;
            case 2: cadena="2"+cadena; break;
            case 3: cadena="3"+cadena; break;
            case 4: cadena="4"+cadena; break;
            case 5: cadena="5"+cadena; break;
            case 6: cadena="6"+cadena; break;
            case 7: cadena="7"+cadena; break;
            case 8: cadena="8"+cadena; break;
            case 9: cadena="9"+cadena; break;
        }
        entero /= 10;
    }
}
```

```
#include "video.h"
#include <SDL.h>
#include <SDL_opengl.h>

void Video::loadSurface(char* surf){
    if ( (surface = SDL_LoadBMP(surf)) ) {

        // Check that the image's width is a power of 2
        if ( (surface->w & (surface->w - 1)) != 0 ) {
            printf("ERROR en imagen, el ancho no es potencia de 2\n");
        }

        // Also check if the height is a power of 2
        if ( (surface->h & (surface->h - 1)) != 0 ) {
            printf("ERROR en imagen, el alto no es potencia de 2\n");
        }

        // get the number of channels in the SDL surface
        nOfColors = surface->format->BytesPerPixel;
        if (nOfColors == 4)      // contains an alpha channel
        {
            if (surface->format->Rmask == 0x000000ff)
                texture_format = GL_RGBA;
            else
                texture_format = GL_BGRA;
        } else if (nOfColors == 3)      // no alpha channel
        {
            if (surface->format->Rmask == 0x000000ff)
                texture_format = GL_RGB;
            else
                texture_format = GL_BGR;
        } else {
            printf("warning: the image is not truecolor.. this will probably break\n");
            // this error should not go unhandled
        }
    }

    // Have OpenGL generate a texture object handle for us
    glGenTextures( 1, &texture );

    // Bind the texture object
    glBindTexture( GL_TEXTURE_2D, texture );

    // Set the texture's stretching properties
    glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR );
    glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR );

    // Edit the texture object's image data using the information SDL_Surface gives us
    glTexImage2D( GL_TEXTURE_2D, 0, nOfColors, surface->w, surface->h, 0,
                  texture_format, GL_UNSIGNED_BYTE, surface->pixels );
}

else {
    printf("SDL could not load image.bmp: %s\n", SDL_GetError());
    SDL_Quit();
}

if ( surface ) {
    SDL_FreeSurface( surface );
}
textures.push_back(texture);
}

void Video::loadSurfaces(){
    loadSurface("ima/mapa/ball.bmp"); //0
    loadSurface("ima/mapa/punto_fijo.bmp"); //1
    loadSurface("ima/mapa/rocket.bmp"); //2
    loadSurface("ima/mapa/metal.bmp"); //3
    loadSurface("ima/mapa/platform.bmp"); //4
    loadSurface("ima/mapa/rope.bmp"); //5
    loadSurface("ima/mapa/lona.bmp"); //6
    loadSurface("ima/mapa/rocket.bmp"); //7
}
```

```
loadSurface("ima/mapa/llegada.bmp"); // 8
loadSurface("ima/mapa/ballbright.bmp"); //9

}

Video::Video(const Config& config)
{
    int w = config.read<int>("Video.Width");
    int h = config.read<int>("Video.Height");
    int depth = config.read<int>("Video.Depth");

    SDL_InitSubSystem(SDL_INIT_VIDEO);
    SDL_GL_SetAttribute(SDL_GL_DOUBLEBUFFER, 1);
    SDL_SetVideoMode(w, h, depth, SDL_OPENGL); // | SDL_FULLSCREEN);

    SDL_WM_SetCaption("Tatu Carreta RUN", 0);

    double ar = static_cast<double>(w) / h;
    glOrtho(-ar, ar, -1.0, 1.0, -1.0, 1.0);
    glEnable(GL_TEXTURE_2D);

    glClearColor( 0.0f, 0.0f, 0.0f, 0.0f );
//    glViewport( 0, 0, w, h );
    glClear( GL_COLOR_BUFFER_BIT );
    glMatrixMode( GL_PROJECTION );
    glLoadIdentity();

//    glMatrixMode( GL_MODELVIEW );
//    glLoadIdentity();

    loadSurfaces();
    glClearColor(1.0f, 1.0f, 1.0f, 0.0f);
}

Video::~Video()
{
    //SDL_QuitSubSystem(SDL_INIT_VIDEO);
}
```

```
#include "zonaLlegada.h"

ZonaLlegada::ZonaLlegada(double x, double y, double width, double height){
    this->x = x;
    this->y = y;
    this->width = width;
    this->height = height;
}
double ZonaLlegada::getX(){
    return x;
}
double ZonaLlegada::getY(){
    return y;
}
double ZonaLlegada::getWidth(){
    return width;
}
double ZonaLlegada::getHeight(){
    return height;
}
```