LexRis Logic Headers V2.00
Guía del Programador

LexRis Logic

1 Guía Inicial

Las Cabeceras de LexRis Logic brindan un código amigable orientado a objetos y funciones en C++ de Matemática Básica, Temporizador, Conversiones de Tipo, Encriptador, Lector de Archivos, Separador de Cadenas, Estructuras Matemáticas, Estructuras de Datos y en especial de las librerías:

- Allegro v5.2.0
- ENet v1.3.13
- irrKlang v1.5.0 (API)

2 namespace LL_AL5

Espacio de nombres que incluye código amigable de la librería para programación de videojuegos: Allegro 5.

2.1 Allegro 5 (LexRisLogic/Allegro5/Allegro5.h)

2.1.1 Definición de Tipo: Posición.

typedef float Type_pos
Tipo de dato usado para indicar posiciones en el eje x e y de todos los objetos a poderse dibujar en pantalla.

2.1.2 Definición de Tipo: Tamaño de Pantalla.

typedef unsigned int Type_display_size
Tipo de dato usado para indicar valores de tamaño de pantalla objetivo o real.

2.1.3 Variable: extern float bitmap_scale_x

Variable que contiene el valor actual de la escala de los mapas de bits en el eje x para ejecutar las operaciones de dibujo.

2.1.4 Variable: extern float bitmap_scale_y

Variable que contiene el valor actual de la escala de los mapas de bits en el eje y para ejecutar las operaciones de dibujo.

2.1.5 Variable: extern float text_scale

Variable que contiene el valor actual de la escala de las fuentes de texto para ejecutar las operaciones de dibujo.

2.1.6 Variable: extern float primitives_scale

Variable que contiene el valor actual de la escala de las figuras primitivas para ejecutar las operaciones de dibujo.

2.1.7 Variable: extern Type_display_size desktop_size_x

Variable que contiene el valor del tamaño de pantalla del escritorio en el eje x que es actualizado cuando el sistema Allegro es iniciado, por defecto empieza con valor de 640.

2.1.8 Variable: extern Type_display_size desktop_size_y

Variable que contiene el valor del tamaño de pantalla del escritorio en el eje y que es actualizado cuando el sistema Allegro es iniciado, por defecto empieza con valor de 480.

2.1.9 Función: Iniciar Sistema Allegro.

void init_allegro()
Inicia el sistema Allegro y actualiza el tamaño de la pantalla de escritorio si se encuentra disponible.

2.1.10 Función: Pausar.

void sleep(float sleep_time)
Pausa la aplicación durante un total de segundos pasado por parámetro.

2.2 Sistema de Archivos (LexRisLogic/Allegro5/Path.h)

2.2.1 Función: Obtener Directorio Actual.

std::string get_current_directory()
Retorna el directorio actual donde se esta ejecutando el programa.

2.2.2 Función: Cambiar Directorio de Ejecución.

bool change_directory(std::string new_path)
Cambia el directorio donde se ejecuta el programa, retorna verdadero en caso de éxito.

2.2.3 Función: Crear Directorio.

bool make_directory(std::string new_dir)
Crea un directorio nuevo de acuerdo a la ruta pasada por parámetro, retorna verdadero en caso de éxito.

2.2.4 Función: Revisar Existencia de una Ruta.

bool path_exists(std::string path)
Retorna verdadero si la ruta pasada por parámetro existe en el sistema de archivos.

2.2.5 Función: Remover Ruta.

bool remove_path(std::string path)
Elimina completamente la ruta pasada por parámetro del sistema de archivos, puede eliminar con éxito archivos y carpetas vacías.

2.3 Cuadros de Diálogo (LexRisLogic/Allegro5/NativeDialog.h)

2.3.1 Función: Iniciar Complemento Cuadros de Diálogo Nativo.

bool native_dialog_addon()
Inicia el complemento de cuadros de diálogo nativo para su uso completo, retorna verdadero en caso de éxito.

2.3.2 Función: Mostrar Cuadro de Diálogo.

bool show_native_message(ALLEGRO_DISPLAY* display, std::string title, std::string header, std::string message, int flag)
Muestra un cuadro de diálogo bloqueando el display enviado solo si es diferente de nulo, donde la cadena *title* es el título del cuadro, la cadena *header* es el título del mensaje, la cadena *message* es el mensaje a mostrar al usuario y el *flag* es el tipo de mensaje que puede ser:

1. **ALLEGRO_MESSAGEBOX_WARN**
   Muestra un mensaje de tipo alerta.
2. **ALLEGRO_MESSAGEBOX_ERROR**
   Muestra un mensaje de tipo error.
3. **ALLEGRO_MESSAGEBOX_QUESTION**
   Muestra un mensaje simple.
4. **ALLEGRO_MESSAGEBOX_OK_CANCEL**
   Muestra un cuadro con opciones de aceptar o cancelar.
5. **ALLEGRO_MESSAGEBOX_YES_NO**
   Muestra un cuadro con opciones de si o no.

Retorna verdadero en caso de haber recibido como respuesta si o aceptar, y falso si la ventana a recibido alguna cancelación, una respuesta de no o cancelar, o fue cerrada por el usuario.

2.3.3 Clase: FileChooser

Clase que se encarga de crear un explorador de archivos donde el usuario indicara la ruta del o los archivos seleccionados.
Funciones:

- **void set_display(ALLEGRO_DISPLAY* display)**
  Asigna la pantalla a bloquearse cuando se abra el explorador de archivos, puede ser nulo.
- **ALLEGRO_DISPLAY* get_display()**
  Retorna un puntero a la pantalla que será bloqueada por el explorador de archivos.
- **void set_initial_path(std::string new_search_path)**
  Indica la ruta inicial del explorador de archivos por donde empezará la búsqueda.
- **std::string get_initial_path()**
  Retorna la ruta inicial del explorador de archivos.
- **void set_title(std::string new_title)**
  Cambia el título del explorador de archivos.
- **std::string get_title()**
  Retorna el título del explorador de archivos.
- **void set_patterns(std::string new_patterns)**
  Cambia la lista de archivos admitidos (patrones), cada patrón debe estar separado por un *;*.

- **std::string get_patterns()**
  Retorna los patrones que serán permitidos por el explorador de archivos.
- **void set_mode(int new_mode)**
  Indica el modo del explorador de archivos y su finalidad con los siguientes flags o combinaciones:
  1. **ALLEGRO_FILECHOOSER_FILE_MUST_EXIST**
     Permite una visualización simple de búsqueda de un archivo.
  2. **ALLEGRO_FILECHOOSER_SAVE**
     El explorador entra en el modo de salvado de un archivo mediante la entrada de un nuevo nombre para el archivo.
  3. **ALLEGRO_FILECHOOSER_FOLDER**
     Crea un explorador de archivos para búsqueda de carpetas.
  4. **ALLEGRO_FILECHOOSER_PICTURES**
     Permite un explorador con visualización de imágenes.
  5. **ALLEGRO_FILECHOOSER_SHOW_HIDDEN**
     Permite la visualización de archivos ocultos.
  6. **ALLEGRO_FILECHOOSER_MULTIPLE**
     Habilita la selección de varios archivos en el explorador de archivos.
- **bool start_filechooser()**
  Inicia el explorador de archivos, retorna verdadero si se consiguió ruta alguna.
- **unsigned int get_number_of_selected_files()**
  Retorna el número de rutas conseguidas.

Operadores:

- **std::string operator [] (unsigned int index)**
  Obtiene una ruta que consiguió el explorador de archivos.

Destructores:

- **~ FileChooser()**
  Destruye la interfaz creada para el explorador de archivos.

2.3.4 Clase: TextLog

Clase que crea un ventana de registro de texto.
Funciones:

- **void set_title(std::string new_title)**
  Cambia el título de la ventana de registro.
- **std::string get_title()**
  Retorna el título del Registro.
- **void set_mode(int new_mode)**
  Indica el comportamiento de la ventana de registro donde se puede combinar los siguientes flags:
  1. **ALLEGRO_TEXTLOG_NO_CLOSE**
     Evita que la ventana del registro genere eventos de cierre.
  2. **ALLEGRO_TEXTLOG_MONOSPACE**
     La ventana de registro usara una fuente de espacio sencillo para mostrar el texto.
- **bool open_textlog()**
  Abre la ventana de registro, retorna verdadero en caso de éxito.

- **bool write_text(std::string text)**
  Escribe una cadena en el registro, retorna verdadero en caso de éxito.
- **bool write_endl()**
  Escribe un salto de línea en el registro, retorna verdadero en caso de éxito.
- **bool close_textlog()**
  Cierra la ventana de registro, retorna verdadero en caso de éxito.
- **bool is_open()**
  Retorna verdadero si la ventana de registro esta abierta.

Operadores:

- **operator ALLEGRO_TEXTLOG*** ()
  Retorna un puntero **ALLEGRO_TEXTLOG** para que pueda ser usado con las funciones restantes de Allegro.

Destructores:

- ∼ **TextLog**()
  Cierra y destruye la interfaz creada para la ventana de registro.

2.4 Pantalla (LexRisLogic/Allegro5/Display.h)

2.4.1 Clase: Display

Clase que provee toda una interfaz para manejar pantallas objetivos de operaciones de dibujo.

Constructores:

- **Display(Type_display_size size_x, Type_display_size size_y)**
  Crea una pantalla de tamaño *size_x*size_y* para ser objetivo de las operaciones de dibujo, el tamaño en el que trabajaran las posiciones sera igual al tamaño de la pantalla y las escalas globales serán normales.
- **Display(Type_display_size size_x, Type_display_size size_y, Type_display_size real_size_x, Type_display_size real_size_y)**
  Crea una pantalla de tamaño *size_x*size_y* para ser objetivo de las operaciones de dibujo, el tamaño en el que trabajaran las posiciones sera igual a una pantalla de tamaño *real_size_x*real_size_y*, las escalas globales serán actualizadas para trabajar con este tamaño de trabajo (Tamaño Real).

Funciones:

- **void set_title(std::string new_title)**
  Cambia el título de la pantalla creada.
- **Type_display_size get_size_x()**
  Retorna el tamaño del eje x de la pantalla.
- **Type_display_size get_size_y()**
  Retorna el tamaño del eje y de la pantalla.
- **void set_real_size(Type_display_size real_size_x,Type_display_size real_size_y)**
  Cambia el tamaño en el que se trabaja actualizando la posición de la cámara y las escalas globales.
- **Type_display_size get_real_size_x()**
  Retorna el tamaño del eje x en el que se trabaja.
- **Type_display_size get_real_size_y()**
  Retorna el tamaño del eje y en el que se trabaja.
- **void set_flag(int new_flag)**
  Un flag indica de que tipo sera la pantalla, al ser modificado el flag, la pantalla se volverá a crear con los nuevos parámetros, este flag toma combinaciones de las macros de Allegro:

  1. **ALLEGRO_WINDOWED**
     Para una pantalla de tamaño fijo en forma de ventana.
  2. **ALLEGRO_FULLSCREEN**
     Utiliza el modo pantalla completa del sistema operativo, usar mas de una actualización seguida en este modo puede causar que el renderizado no sea óptimo.
  3. **ALLEGRO_FULLSCREEN_WINDOW**
     Genera una pantalla completa en forma de ventana, no genera errores con la actualización seguida.
  4. **ALLEGRO_RESIZABLE**
     Solo combinable con **ALLEGRO_WINDOWED**, logra que el tamaño de la pantalla pueda ser manejado por el usuario, activando la opción maximizar.

- **void resize(Type_display_size size_x, Type_display_size size_y)**
  Redimensiona la pantalla a tamaño *size_x*size_y* actualizando las escalas globales con respecto al tamaño real, solo se puede usar en modo ALLEGRO_WINDOWED o ALLEGRO_RESIZABLE.
- **void set_cam(Type_pos new_pos_x, Type_pos new_pos_y)**
  Cambia la posición de la cámara, será visible todo lo dibujado en pantalla desde las coordenadas (x,y) hasta el tamaño real en el que se trabaja.
- **void set_cam_x(Type_pos new_pos_x)**
  Cambia la posición de la cámara en el eje x.
- **Type_pos get_cam_x()**
  Retorna el posición de la cámara en el eje x.
- **void set_cam_y(Type_pos new_pos_y)**
  Cambia la posición de la cámara en el eje y.
- **Type_pos get_cam_y()**
  Retorna el posición de la cámara en el eje y.
- **void plus_cam_x(Type_pos plus_x)**
  Aumenta la posición de la cámara en el eje x.
- **void plus_cam_y(Type_pos plus_y)**
  Aumenta la posición de la cámara en el eje y.
- **Type_pos display_to_real_pos_x(Type_pos pos_x,bool references_axes=true)**
  Transforma la posición x pasada por parámetro que toma como referencia al tamaño de la pantalla y la transforma a una coordenada real, es decir, a una coordenada en la que se está trabajando; el segundo parámetro es para tomar en cuenta la posición de la cámara.
- **Type_pos display_to_real_pos_y(Type_pos pos_y,bool references_axes=true)**
  Transforma la posición y pasada por parámetro que toma como referencia al tamaño de la pantalla y la transforma a una coordenada real, es decir, a una coordenada en la que se está trabajando; el segundo parámetro es para tomar en cuenta la posición de la cámara.
- **bool show_cursor()**
  Vuelve visible al cursor del ratón en la pantalla.
- **bool hide_cursor()**
  Oculta el cursor del ratón en la pantalla.
- **bool set_cursor(ALLEGRO_BITMAP* bitmap_cursor,**
  int focus_x,int focus_y)
  Asigna un mapa de bits al cursor del ratón indicando que punto de la imagen es el punto de enfoque del ratón.
- **bool destroy_cursor()**
  Destruye y libera la memoria utilizada por el cursor del ratón, retorna verdadero en caso de éxito.
- **void clear()**
  Limpia la pantalla con el color blanco por defecto.
- **void clear_to_color(ALLEGRO_COLOR color)**
  Limpia la pantalla con el color enviado por parámetro.

- **template<typename T>**
  void draw(T* object,bool references_axes=true)
  Función que recibe una clase **T** que debe tener los siguientes métodos implementados:

  1. **Type_pos get_pos_x()** → Usado para obtener la posición x del objeto teniendo en cuenta que trabaja conforme al tamaño real que se ha asignado.
  2. **Type_pos get_pos_y()** → Usado para obtener la posición y del objeto teniendo en cuenta que trabaja conforme al tamaño real que se ha asignado.
  3. **void set_pos(Type_pos new_pos_x,Type_pos new_pos_y)** → Usado para asignar la verdadera posición conforme al tamaño real que se ha asignado y al tamaño de la pantalla objetivo, terminada la operación de dibujo se reasignara su verdadera posición.
  4. **void draw()** → Usado para dibujar al objeto, tener en cuenta que su posición será cambiada y aprovechar las escalas globales para una correcta operación de dibujo.

  La operación de dibujo puede ser controlada con el parámetro references_axes para indicar si se tomará en cuenta la posición de la cámara, por defecto siempre se toma en cuenta esta posición.

- **void refresh()**
  Actualiza y muestra la imagen generada en pantalla.
- **void set_target()**
  Asigna la pantalla como objetivo de las operaciones de dibujo.

Operadores:

- **operator ALLEGRO_DISPLAY* ()**
  Retorna un puntero **ALLEGRO_DISPLAY** para que pueda ser usado con las funciones restantes de Allegro.
- **operator ALLEGRO_BITMAP* ()**
  Retorna una mapa de bits especial representando al buffer de la pantalla, tener mucho cuidado con el uso de otras operaciones de Allegro con este mapa de bits.
- **operator ALLEGRO_MOUSE_CURSOR* ()**
  Retorna un puntero **ALLEGRO_MOUSE_CURSOR** para que pueda ser usado con las funciones restantes de Allegro.

Destructores:

- **~ Display()**
  Destruye la pantalla y el cursor del ratón liberando la memoria utilizada.

2.5 Temporizador (LexRisLogic/Allegro5/Timer.h)

2.5.1 Clase: Timer

Clase que representa a un temporizador que genera eventos periódicamente.
Funciones:

- **bool set_speed_seconds(double new_speed_seconds)**
  Asigna el periodo de generación de eventos en segundos, retorna verdadero en caso de éxito.
- **double get_speed_seconds(double new_speed_seconds)**
  Retorna el valor del periodo de generación de eventos en segundos.
- **bool create()**
  Crea el temporizador inicialmente detenido.
- **bool destroy()**
  Destruye el temporizador, tener cuidado cuando se asigna el temporizador a los eventos.
- **bool start()**
  Inicia el temporizador empezando a generar eventos.
- **bool resume()**
  Resume el conteo de generación de eventos del temporizador.
- **bool stop()**
  Detiene la generación de eventos del temporizador.

Operadores:

- **operator ALLEGRO_TIMER*** ()
  Retorna un puntero **ALLEGRO_TIMER** para que pueda ser usado con las funciones restantes de Allegro.

Destructores:

- ∼ **Timer**()
  Destruye el temporizador liberando la memoria utilizada.

2.6 Entradas (LexRisLogic/Allegro5/Input.h)

2.6.1 Clase: KeyControl

Clase que representa una estructura básica para controlar los eventos de teclas por llaves.

Funciones:

- **bool add_key(std::string key_name,int keycode)**
  Inserta una llave a la estructura asignándole un nombre y su código de tecla.
- **bool find_key(std::string key_name)**
  Busca si existe una llave con el nombre enviado por parámetro, retorna verdadero en caso de éxito.
- **bool find_key(int keycode,std::string* key_name=nullptr)**
  Busca si existe una llave con el código de tecla enviado por parámetro, retorna verdadero en caso de éxito; puede obtenerse el nombre de la llave que tiene asignado el código de tecla enviando un puntero a una cadena.
- **bool mod_key(std::string last_key_name,std::string new_key_name)**
  Cambia el nombre de una llave con otro, retorna verdadero en caso de éxito.
- **bool mod_key(std::string key_name,int new_keycode)**
  Cambia el código de tecla de la llave.
- **bool remove_key(std::string key_name)**
  Remueve la llave que tenga el nombre enviado por parámetro.
- **bool remove_key(int keycode)**
  Remueve la llave que tenga el código de tecla enviado por parámetro.
- **unsigned int size()**
  Retorna el número de llaves que están en la estructura.
- **void clear_key_status()**
  Cambia el estado de todas las llaves a falso, es decir, sin presionar.
- **void clear()**
  Vacía la estructura de llaves.
- **bool& get_key_down_status(std::string key_name)**
  Obtiene el estado de la llave que tenga el nombre pasado por parámetro, retorna por referencia para su modificación, si la llave no existe retorna un comodín evitando errores, revisar el nombre de las llaves cuando se use esta estructura.
- **int get_keycode(std::string key_name)**
  Retorna el código de tecla que se asigno a una llave que tiene el nombre pasado por parámetro.

Destructores:

- **~ KeyControl()**
  Destruye la estructura y libera la memoria utilizada.

2.6.2 Clase: Input

Clase que provee una interfaz para manejar la mayoría de eventos que nos permite controlar Allegro como el ratón, teclado, pantallas, ventanas de registro y temporizadores.
Constructores:

- **Input**()
  Instala el sistema para manejo del ratón y del teclado, creando también la cola de eventos.

Funciones:

- **bool unregister_timer()**
  Anula el registro del temporizador, desactiva el control de eventos del temporizador, retorna verdadero en caso de éxito.
- **bool register_timer(ALLEGRO_TIMER* new_timer)**
  Registra un temporizador si no hay uno ya registrado, activa el control de eventos del temporizador, retorna verdadero en caso de éxito.
- **bool unregister_display()**
  Anula el registro de la pantalla y desactiva el control de eventos de la pantalla, retorna verdadero en caso de éxito.
- **bool register_display(ALLEGRO_DISPLAY* new_display)**
  Registra una pantalla solo si no hay una ya registrada, activa el control de eventos de la pantalla, retorna verdadero en caso de éxito.
- **bool unregister_textlog()**
  Anula el registro de la ventana de registro, desactiva el control de eventos de la ventana de registro, retorna verdadero en caso de éxito.
- **bool register_textlog(ALLEGRO_TEXTLOG* new_textlog)**
  Registra una ventana de registro si no hay una ya registrada, activa el control de eventos de la ventana de registro, retorna verdadero en caso de éxito.
- **void set_key_control(KeyControl* new_key_control)**
  Establece la estructura de llaves a utilizar, se envía el puntero para evitar una copia de la estructura interna.
- **KeyControl* get_key_control()**
  Retorna un puntero a la estructura de llaves que se esta usando actualmente.
- **void clear_events()**
  Vacía la cola de eventos, si la cola de eventos esta activa y no esta siendo controlada, empezara a generar eventos en tiempos no deseados.
- **void clear_key_status()**
  Establece el estado de cada llave de la estructura de llaves utilizada actualmente a falso.
- **bool set_wait_time(float wait_time)**
  Asigna el tiempo de espera de generación de eventos.
- **float get_wait_time()**
  Retorna el tiempo de espera asignado para la generación de eventos, por defecto la espera es 0, un tiempo inmediato.
- **bool keyboard_on()**
  Registra los eventos del teclado.
- **bool keyboard_off()**
  Anula el registro del teclado, y deja de controlar sus eventos generados.

- **bool input_on(std::string* input_objetive,unsigned int max_input_size, bool special_is_blocked=false)**
  Activa la entrada de una cadena mediante eventos, el cual se le asigna un tamaño máximo válido y la cadena generada se guardara en el contenido de la cadena objetivo, la variable final indica si se bloqueara la lectura para caracteres especiales como el salto de línea $'\backslash n'$ y tabulaciones $'\backslash t'$.
- **bool input_off(std::string* input_objetive)**
  Desactiva la entrada de la cadena mediante eventos solo si se envía la cadena objetivo para poder asegurar la existencia del mismo.
- **bool mouse_on(bool event_type=false)**
  Activa la obtención de datos del ratón que puede ser mediante eventos u obtención directa de los datos, el parámetro enviado define la forma de obtención, por defecto es sin eventos por la gran cantidad de eventos generados por el ratón.
- **bool mouse_off()**
  Anula el registro del ratón, y deja de controlar sus eventos generados si fue activada esta forma de obtención.
- **bool set_mouse_xy(int new_mouse_x,int new_mouse_y)**
  Mueve la posición del ratón a la posición (x,y) en la pantalla registrada, retorna verdadero en caso de éxito.
- **int get_mouse_x()**
  Retorna la posición actual del ratón en el eje x en la pantalla registrada.
- **int get_mouse_y()**
  Retorna la posición actual del ratón en el eje y en la pantalla registrada.
- **bool set_mouse_z(int new_mouse_z)**
  Cambia el valor del eje de la rueda del ratón, retorna verdadero en caso de en éxito.
- **int get_mouse_z()**
  Retorna la posición actual del eje de la rueda del ratón.
- **bool& left_click()**
  Retorna el estado del botón izquierdo del ratón, el valor puede ser modificado.
- **bool& right_click()**
  Retorna el estado del botón derecho del ratón, el valor puede ser modificado.
- **bool& middle_click()**
  Retorna el estado del botón central del ratón, el valor puede ser modificado.
- **bool get_timer_event()**
  Retorna el estado del temporizador, verdadero si el evento generado le pertenece al ratón.
- **bool& get_display_status()**
  Retorna el estado de la pantalla registrada, verdadero si se ha generado el evento de salida de la pantalla, el valor puede ser modificado.
- **bool& get_textlog_status()**
  Retorna el estado de la ventana de registro, verdadero si se ha generado el evento de salida de la ventana de registro, el valor puede ser modificado.
- **bool get_event()**
  Consigue los eventos registrados añadiéndolos en la cola de eventos, retorna verdadero si se llego a obtener un evento en el tiempo de espera.
- **int get_keycode()**
  Obtiene el código de tecla de un evento de teclado al presionar la tecla, es necesario tener una pantalla registrada para obtener los eventos del teclado, si se cierra la pantalla retorna -1.

Operadores:

- **bool& operator [] (std::string key_name)**
  Retorna el estado de una llave con el nombre pasado por parámetro, si se encuentra en la estructura de llaves asignada se retorna por referencia su valor para que pueda ser modificado, retorna un comodín si no se encuentra.
- **operator ALLEGRO_EVENT_QUEUE* ()**
  Retorna un puntero **ALLEGRO_EVENT_QUEUE** para que pueda ser usado con las funciones restantes de Allegro.

Destructores:

- **~ Input()**
  Anula el registro de la pantalla, ventana de registro, temporizador, teclado, ratón, destruyendo la cola de eventos y desinstala el sistema del manejo del ratón y del teclado.

2.7 Color (LexRisLogic/Allegro5/Color.h)

2.7.1 Estructura: Color

Estructura básica de un color.
Constructores:

- **Color(unsigned char** new_red=0,**unsigned char** new_green=0,
  unsigned char new_blue=0,**unsigned char** new_alpha=255)
  Crea un color pasando su código en RGB y el nivel de opacidad, por defecto es un color negro completamente opaco.
- **Color(ALLEGRO_COLOR** color)
  Crea un color a partir de la estructura **ALLEGRO_COLOR** de Allegro.

Variables:

- **unsigned char red=0**
  Representa a la cantidad de color Rojo, valor entre 0 a 255.
- **unsigned char green=0**
  Representa a la cantidad de color Verde, valor entre 0 a 255.
- **unsigned char blue=0**
  Representa a la cantidad de color Azul, valor entre 0 a 255.
- **unsigned char alpha=255**
  Representa a la cantidad de opacidad del color, valor entre 0 a 255.

Operadores:

- **Color operator ! ()**
  Retorna el color opuesto en síntesis aditiva, pero con la misma opacidad.
- **Color operator = (ALLEGRO_COLOR another_color)**
  Copia el color de un **ALLEGRO_COLOR** a esta estructura.
- **bool operator == (Color another_color)**
  Compara la estructura con otra retornando verdadero si tienen el mismo color e igual opacidad.
- **bool operator != (Color another_color)**
  Compara la estructura con otra retornando verdadero si tienen diferente color o diferente opacidad.
- **operator ALLEGRO_COLOR ()**
  Retorna una estructura **ALLEGRO_COLOR** para que pueda ser utilizado con las funciones restantes de Allegro.

2.8 Primitivos (LexRisLogic/Allegro5/Primitives.h)

2.8.1 Función: Iniciar Complemento Primitivos.

bool primitives_addon()
Inicia el complemento de figuras primitivas para su uso completo, retorna verdadero en caso de éxito.

2.8.2 Función: Detener Complemento Primitivos.

void uninstall_primitives()
Detiene el complemento de primitivos iniciado anteriormente, por defecto siempre se llama cuando termina la ejecución del programa.

2.8.3 Clase: Pixel

Primitivo que consiste en un pixel de la pantalla.
Constructores:

- **Pixel**()
  Inicia los valores del pixel en la posición (0,0).
- **Pixel**(**Type_pos** pos_x,**Type_pos** pos_y)
  Inicia los valores del pixel en la posición (pos_x,pos_y).

Funciones:

- **void set_pos(Type_pos new_pos_x,Type_pos new_pos_y)**
  Cambia la posición del Pixel a (new_pos_x,new_pos_y).
- **void set_pos_x(Type_pos new_pos_x)**
  Cambia la posición del pixel en el eje x.
- **Type_pos get_pos_x()**
  Retorna la posición actual del pixel en el eje x.
- **void set_pos_y(Type_pos new_pos_y)**
  Cambia la posición del pixel en el eje y.
- **Type_pos get_pos_y()**
  Retorna la posición actual del pixel en el eje y.
- **void set_color(ALLEGRO_COLOR new_color)**
  Cambia el color del pixel.
- **ALLEGRO_COLOR get_color()**
  Retorna el color actual del pixel.
- **void draw()**
  Dibuja el pixel en su respectiva posición y su color tomando en cuenta las escalas globales.
- **void draw_in_another_target()**
  Dibuja el pixel sin tomar en cuenta las escalas globales, usado para dibujar en otro objetivo.

2.8.4 Clase: Line

Primitivo que consiste en un segmento de línea.
Constructores:

- **Line**()
  Inicia los valores del segmento de línea con (0,0) como posición inicial y (0,0) como posición final.
- **Line**(**Type_pos** pos_x1,**Type_pos** pos_y1,**Type_pos** pos_x2,**Type_pos** pos_y2)
  Inicia los valores del segmento de línea con (pos_x1,pos_y1) como posición inicial y (pos_x2,pos_y2) como posición final.

Funciones:

- **void set_pos(Type_pos new_cam_pos_x,Type_pos new_cam_pos_y)**
  Cambia la posición que se toma por referencia para la cámara a (new_cam_pos_x,new_cam_pos_y).
- **void set_pos_x(Type_pos new_cam_pos_x)**
  Cambia la posición en el eje x que se toma por referencia para la cámara.
- **Type_pos get_pos_x()**
  Retorna la posición tomada por referencia para la cámara en el eje x.
- **void set_pos_y(Type_pos new_cam_pos_y)**
  Cambia la posición en el eje y que se toma por referencia para la cámara.
- **Type_pos get_pos_y()**
  Retorna la posición tomada por referencia para la cámara en el eje y.
- **void set_points(Type_pos new_pos_x1,Type_pos new_pos_y1, Type_pos new_pos_x2,Type_pos new_pos_y2)**
  Cambia la posición del punto inicial y final del segmento.
- **void set_pos_1(Type_pos new_pos_x1,Type_pos new_pos_y1)**
  Cambia la posición del punto inicial del segmento.
- **Type_pos get_pos_x1()**
  Retorna la posición del eje x del punto inicial.
- **Type_pos get_pos_y1()**
  Retorna la posición del eje y del punto inicial.
- **void set_pos_2(Type_pos new_pos_x2,Type_pos new_pos_y2)**
  Cambia la posición del punto final del segmento.
- **Type_pos get_pos_x2()**
  Retorna la posición del eje x del punto final.
- **Type_pos get_pos_y2()**
  Retorna la posición del eje y del punto final.
- **void set_color(ALLEGRO_COLOR new_color)**
  Cambia el color del segmento.
- **ALLEGRO_COLOR get_color()**
  Retorna el color actual del segmento.
- **void set_thickness(float new_thickness)**
  Cambia el grosor del segmento.
- **float get_thickness()**
  Retorna el grosor actual del segmento.
- **void draw()**
  Dibuja el segmento tomando en cuenta la posición de la cámara como referencia, su posición real asignada y las escalas globales.
- **void draw_in_another_target()**
  Dibuja el segmento sin tomar en cuenta las escalas globales, usado para dibujar en otro objetivo.

2.8.5 Clase Base: Primitive

Clase Base que contiene la estructura base de una figura primitiva.
Funciones:

- **void set_thickness(float new_thickness)**
  Cambia el grosor de la primitiva.
- **float get_thickness()**
  Retorna el grosor actual de la primitiva.
- **void set_color(ALLEGRO_COLOR new_color)**
  Cambia el color de la primitiva.
- **ALLEGRO_COLOR get_color()**
  Retorna el color actual de la primitiva.
- **void set_filled_status(bool is_filled)**
  Cambia la opción de relleno de la primitiva.
- **bool get_filled_status()**
  Retorna el estado de la opción de relleno de la primitiva.

2.8.6 Clase Base: Figure

Clase Base que hereda de la clase base **Primitive** aumentando en la estructura base la posición de una figura común.
Funciones:

- **void set_pos(Type_pos new_pos_x,Type_pos new_pos_y)**
  Cambia la posición de la primitiva a (new_pos_x,new_pos_y).
- **void set_pos_x(Type_pos new_pos_x)**
  Cambia la posición de la primitiva en el eje x.
- **Type_pos get_pos_x()**
  Retorna la posición actual de la primitiva en el eje x.
- **void set_pos_y(Type_pos new_pos_y)**
  Cambia la posición de la primitiva en el eje y.
- **Type_pos get_pos_y()**
  Retorna la posición actual de la primitiva en el eje y.

2.8.7 Clase: Circle

Clase que hereda de la clase base **Figure** representando a la figura círculo.
Constructores:

- **Circle**()
  Inicia los valores del círculo en la posición (0,0) con radio 1.
- **Circle**(**Type_pos** pos_x,**Type_pos** pos_y,**float** radius)
  Inicia los valores del círculo en la posición (pos_x,pos_y) con el radio pasado por parámetro.

Funciones:

- **void set_radius(float new_radius)**
  Cambia el radio del círculo.
- **float get_radius()**
  Retorna el radio actual del círculo.
- **void draw()**
  Dibuja el círculo tomando en cuenta su posición y las escalas globales para el tamaño real del radio.
- **void draw_in_another_target()**
  Dibuja el círculo sin tomar en cuenta las escalas globales, usado para dibujar en otro objetivo.

2.8.8 Clase: Ellipse

Clase que hereda de la clase base **Figure** representando a la figura elipse.
Constructores:

- **Ellipse**()
  Inicia los valores de la elipse en la posición (0,0) con radio 1 en el eje x e y.
- **Ellipse**(**Type_pos** pos_x,**Type_pos** pos_y,**float** radius_x,**float** radius_y)
  Inicia los valores de la elipse en la posición (pos_x,pos_y) con los radios del eje x e y pasados por parámetros.

Funciones:

- **void set_radius_x(float new_radius_x)**
  Cambia el radio del eje x de la elipse.
- **float get_radius_x()**
  Retorna el radio actual del eje x de la elipse.
- **void set_radius_y(float new_radius_y)**
  Cambia el radio del eje y de la elipse.
- **float get_radius_y()**
  Retorna el radio actual del eje y de la elipse.
- **void draw()**
  Dibuja la elipse tomando en cuenta su posición y las escalas globales para el tamaño real de sus radios.
- **void draw_in_another_target()**
  Dibuja la elipse sin tomar en cuenta las escalas globales, usado para dibujar en otro objetivo.

2.8.9 Clase: Rectangle

Clase que hereda de la clase base **Figure** representando a la figura rectángulo.
Constructores:

- **Rectangle**()
  Inicia los valores del rectángulo en la posición (0,0) con tamaño 0 en el eje x e y.
- **Rectangle**(**Type_pos** pos_x,**Type_pos** pos_y,**float** size_x,**float** size_y)
  Inicia los valores del rectángulo en la posición (pos_x,pos_y) con los tamaños del eje x e y pasados por parámetros.

Funciones:

- **void set_size_x(float new_size_x)**
  Cambia el tamaño del eje x del rectángulo.
- **float get_size_x()**
  Retorna el tamaño actual del eje x del rectángulo.
- **void set_size_y(float new_size_y)**
  Cambia el tamaño del eje y del rectángulo.
- **float get_size_y()**
  Retorna el tamaño actual del eje y del rectángulo.
- **void draw()**
  Dibuja el rectángulo tomando en cuenta su posición y las escalas globales para el tamaño real en cada eje.
- **void draw_in_another_target()**
  Dibuja el rectángulo sin tomar en cuenta las escalas globales, usado para dibujar en otro objetivo.

2.8.10 Clase: Triangle

Clase que hereda de la clase base **Primitive** representando a la primitiva triángulo.
Constructores:

- **Triangle**()
  Inicia los valores del triángulo con (0,0) como posición de todos los vértices.
- **Triangle**(**Type_pos** pos_x1,**Type_pos** pos_y1,
  Type_pos pos_x2,**Type_pos** pos_y2,
  Type_pos pos_x3,**Type_pos** pos_y3)
  Inicia los valores del triángulo pasando por parámetro la posición de todos los vértice.

Funciones:

- **void set_pos(Type_pos new_cam_pos_x,Type_pos new_cam_pos_y)**
  Cambia la posición que se toma por referencia para la cámara a (new_cam_pos_x,new_cam_pos_y).
- **void set_pos_x(Type_pos new_cam_pos_x)**
  Cambia la posición en el eje x que se toma por referencia para la cámara.
- **Type_pos get_pos_x()**
  Retorna la posición tomada por referencia para la cámara en el eje x.
- **void set_pos_y(Type_pos new_cam_pos_y)**
  Cambia la posición en el eje y que se toma por referencia para la cámara.
- **Type_pos get_pos_y()**
  Retorna la posición tomada por referencia para la cámara en el eje y.

- **void set_points(Type_pos new_pos_x1,Type_pos new_pos_y1,**
  Type_pos new_pos_x2,Type_pos new_pos_y2,
  Type_pos new_pos_x3,Type_pos new_pos_y3)
  Cambia la posición de todos los vértices del triángulo.
- **void set_pos_1(Type_pos new_pos_x1,Type_pos new_pos_y1)**
  Cambia la posición del primer vértice del triángulo.
- **Type_pos get_pos_x1()**
  Retorna la posición del eje x del primer vértice del triángulo.
- **Type_pos get_pos_y1()**
  Retorna la posición del eje y del primer vértice del triángulo.
- **void set_pos_2(Type_pos new_pos_x2,Type_pos new_pos_y2)**
  Cambia la posición del segundo vértice del triángulo.
- **Type_pos get_pos_x2()**
  Retorna la posición del eje x del segundo vértice del triángulo.
- **Type_pos get_pos_y2()**
  Retorna la posición del eje y del segundo vértice del triángulo.
- **void set_pos_3(Type_pos new_pos_x3,Type_pos new_pos_y3)**
  Cambia la posición del tercer vértice del triángulo.
- **Type_pos get_pos_x3()**
  Retorna la posición del eje x del tercer vértice del triángulo.
- **Type_pos get_pos_y3()**
  Retorna la posición del eje y del tercer vértice del triángulo.
- **void draw()**
  Dibuja el triángulo tomando en cuenta la posición de la cámara como referencia, los vértices asignados y las escalas globales.
- **void draw_in_another_target()**
  Dibuja el triángulo sin tomar en cuenta las escalas globales, usado para dibujar en otro objetivo.

2.8.11 Clase: Function

Clase que representa a una función en dimensión 2 del tipo $y = f(x)$ para visualizarse en pantalla.

Constructores:

- **Function**()
  Inicia el objeto sin asignarle una función para visualizar.
- **Function**(**Type_pos** (*Function_fx)(**Type_pos**))
  Asigna la función enviada por parámetro como la función en dimensión 2 $y = f(x)$ para visualizarse en pantalla, por defecto solo se dibuja una recta de (0,f(0)) a (1,f(1)).

Funciones:

- **void set_pos(Type_pos new_cam_pos_x,Type_pos new_cam_pos_y)**
  Cambia la posición que se toma por referencia para la cámara a (new_cam_pos_x,new_cam_pos_y).
- **void set_pos_x(Type_pos new_cam_pos_x)**
  Cambia la posición en el eje x que se toma por referencia para la cámara.
- **Type_pos get_pos_x()**
  Retorna la posición tomada por referencia para la cámara en el eje x.
- **void set_pos_y(Type_pos new_cam_pos_y)**
  Cambia la posición en el eje y que se toma por referencia para la cámara.

- **Type_pos get_pos_y()**
  Retorna la posición tomada por referencia para la cámara en el eje y.
- **void set_thickness(float new_thickness)**
  Cambia el grosor de las rectas generadas por la función.
- **float get_thickness()**
  Retorna el grosor de las rectas generadas por la función.
- **void set_color(ALLEGRO_COLOR new_color)**
  Cambia el color de las rectas generadas por la función.
- **ALLEGRO_COLOR get_color()**
  Retorna el color actual de las rectas generadas por la función.
- **bool set_step(float new_step)**
  Cambia el paso en el eje x para dibujar las rectas que genere la función, retorna verdadero si el paso es válido.
- **float get_step()**
  Retorna el paso en el eje x que se usa para dibujar la función.
- **void set_initial_x(Type_pos new_init)**
  Cambia la posición inicial en el eje x desde la cual se dibujara la función.
- **Type_pos get_initial_x()**
  Retorna la posición inicial que se toma en cuenta para dibujar la función.
- **void set_final_x(Type_pos new_final)**
  Cambia la posición final en el eje x hasta donde se dibujara la función.
- **Type_pos get_final_x()**
  Retorna la posición final que se toma en cuenta para dibujar la función.
- **void set_function(Type_pos (*Function_fx)(Type_pos))**
  Cambia la función en dimensión 2 $y = f(x)$ para visualizarse en pantalla.
- **void draw()**
  Dibuja la función de acuerdo al paso, desde la posición inicial hasta la posición final con todas las configuraciones hechas tomando en cuenta la posición de la cámara como referencia y las escalas globales.
- **void draw_in_another_target()**
  Dibuja la función sin tomar en cuenta las escalas globales, usado para dibujar en otro objetivo.

2.9 Mapa de Bits (LexRisLogic/Allegro5/Bitmap.h)

2.9.1 Función: Iniciar Complemento Imágenes.

bool image_addon()
Inicia el complemento de imágenes para su uso completo, retorna verdadero en caso de éxito.

2.9.2 Función: Guardar Mapa de Bits.

bool save_bitmap(std::string bitmap_file_name,ALLEGRO_BITMAP* bitmap)
Guarda un mapa de bits en la ruta enviada por parámetro, retorna verdadero en caso de éxito.

2.9.3 Clase Base: BitmapBase

Clase Base que contiene la estructura base de una imagen en pantalla.
Funciones:

- **void set_pos(Type_pos new_pos_x,Type_pos new_pos_y)**
  Cambia la posición del mapa de bits a (new_pos_x,new_pos_y).
- **void set_pos_x(Type_pos new_pos_x)**
  Cambia la posición del mapa de bits en el eje x.
- **Type_pos get_pos_x()**
  Retorna la posición actual del mapa de bits en el eje x.
- **void set_pos_y(Type_pos new_pos_y)**
  Cambia la posición del mapa de bits en el eje y.
- **Type_pos get_pos_y()**
  Retorna la posición actual del mapa de bits en el eje y.
- **void set_angle(float new_angle)**
  Cambia el ángulo de giro del mapa de bits, el ángulo trabaja en radianes.
- **float get_angle()**
  Retorna el ángulo de giro del mapa de bits.
- **void set_flag(int new_flag)**
  Cambia el modo en como se realizara la operación de dibujo del mapa de bits, el modo puede tomar el valor de 0 para una operación por defecto o combinaciones de los siguientes flags de Allegro:
  1. **ALLEGRO_FLIP_HORIZONTAL**
     Dibuja el mapa de bits invirtiendo el eje y.
  2. **ALLEGRO_FLIP_VERTICAL**
     Dibuja el mapa de bits invirtiendo el eje x.
- **void set_scale_x(float new_scale_x)**
  Cambia la escala del mapa de bits en el eje x.
- **float get_scale_x()**
  Retorna la escala del mapa de bits en el eje x.
- **void set_scale_y(float new_scale_y)**
  Cambia la escala del mapa de bits en el eje y.
- **float get_scale_y()**
  Retorna la escala del mapa de bits en el eje y.
- **void set_centering_option(bool new_centering_option_x,bool new_centering_option_y)**
  Cambia la opción de centrado para la operación de dibujo, si la opción es verdadera, se tomara la posición en el eje indicado como central y no como inicial.

- **bool get_centering_option_x()**
  Retorna el estado de la opción de centrado en el eje x.
- **bool get_centering_option_y()**
  Retorna el estado de la opción de centrado en el eje y.

2.9.4 Clase: Bitmap

Clase que hereda de la clase base **BitmapBase** representando a un mapa de bits simple.
Funciones:

- **float get_size_x()**
  Retorna el ancho del mapa de bits.
- **float get_size_y()**
  Retorna el alto del mapa de bits.
- **bool create(int size_x,int size_y)**
  Crea un mapa de bits con el ancho y alto pasados por parámetros, retorna verdadero en caso de éxito.
- **bool destroy()**
  Elimina el mapa de bits, liberando la memoria utilizada, retorna verdadero en caso de éxito.
- **void set_target()**
  Asigna al mapa de bits como objetivo de las operaciones de dibujo.
- **bool lock()**
  Asegura el mapa de bits en memoria para prepararlo contra múltiples accesos de lectura y escritura, retorna verdadero en caso de éxito.
- **void unlock()**
  Quita el bloqueo del mapa de bits liberando la memoria utilizada para los múltiples accesos.
- **ALLEGRO_COLOR get_pixel_color(Type_pos pos_x,Type_pos pos_y)**
  Retorna el color actual que tiene un pixel en el mapa de bits pasando por parámetros su posición.
- **void draw()**
  Dibuja el mapa de bits tomando en cuenta toda la configuración dada y las escalas globales para el tamaño real en cada eje.
- **void draw_in_another_target()**
  Dibuja el mapa de bits sin tomar en cuenta las escalas globales, usado para dibujar en otro objetivo.

Operadores:

- **operator ALLEGRO_BITMAP* ()**
  Retorna un puntero **ALLEGRO_BITMAP** que representa al mapa de bits para que pueda ser usado con las funciones restantes de Allegro.

Destructores:

- **~ Bitmap()**
  Libera la memoria utilizada por la clase y elimina el mapa de bits junto con todos los mapas de bits hijos generados.

2.9.5 Clase: SubBitmap

Clase que hereda de la clase base **BitmapBase** representando a una porción de otro mapa de bits.
Funciones:

- **void set_parent_bitmap(ALLEGRO_BITMAP* new_parent_bitmap)**
  Asigna el mapa de bits padre del cual se tomará porciones de él.
- **ALLEGRO_BITMAP* get_parent_bitmap()**
  Retorna el mapa de bits padre asignado.
- **void set_sub_x(int new_sub_x)**
  Asigna la posición inicial en x a seleccionarse del mapa de bits padre.
- **int get_sub_x()**
  Retorna la posición inicial de selección del mapa de bits en el eje x.
- **void set_sub_y(int new_sub_y)**
  Asigna la posición inicial en y a seleccionarse del mapa de bits padre.
- **int get_sub_y()**
  Retorna la posición inicial de selección del mapa de bits en el eje y.
- **void set_size_x(float new_size_x)**
  Asigna el ancho a seleccionarse del mapa de bits.
- **float get_size_x()**
  Retorna el ancho seleccionado del mapa de bits.
- **void set_size_y(ize new_size_y)**
  Asigna el alto a seleccionarse del mapa de bits.
- **float get_size_y()**
  Retorna el alto seleccionado del mapa de bits.
- **bool create()**
  Crea un mapa de bits a partir de una porción del mapa de bits padre tomando en cuenta la configuración de las posiciones de selección y el ancho y alto a tomarse del mapa de bits, retorna verdadero en caso de éxito.
- **bool destroy()**
  Elimina la porción del mapa de bits, liberando la memoria utilizada, retorna verdadero en caso de éxito.
- **void set_target()**
  Asigna la porción del mapa de bits como objetivo de las operaciones de dibujo.
- **bool lock()**
  Asegura la porción del mapa de bits en memoria para prepararlo contra múltiples accesos de lectura y escritura, retorna verdadero en caso de éxito.
- **void unlock()**
  Quita el bloqueo de la porción del mapa de bits liberando la memoria utilizada para los múltiples accesos.
- **ALLEGRO_COLOR get_pixel_color(Type_pos pos_x,Type_pos pos_y)**
  Retorna el color actual que tiene un pixel en la porción del mapa de bits pasando por parámetros su posición.
- **void draw()**
  Dibuja la porción del mapa de bits tomando en cuenta toda la configuración dada y las escalas globales para el tamaño real en cada eje.
- **void draw_in_another_target()**
  Dibuja la porción del mapa de bits sin tomar en cuenta las escalas globales, usado para dibujar en otro objetivo.

Operadores:

- **operator ALLEGRO_BITMAP*** ()
  Retorna un puntero **ALLEGRO_BITMAP** que representa a la porción del mapa de bits padre para que pueda ser usado con las funciones restantes de Allegro.

Destructores:

- ~ **SubBitmap**()
  Libera la memoria utilizada por la clase y elimina solo el mapa de bits hijo generado.

2.9.6 Clase: Image

Clase que hereda de la clase base **BitmapBase** representando una imagen como un mapa de bits.
Funciones:

- **void set_path(std::string new_image_path)**
  Se asigna la ruta de la imagen a cargarse por la clase.
- **std::string get_path()**
  Retorna la ruta de la imagen a cargarse por la clase.
- **float get_size_x()**
  Retorna el ancho de la imagen.
- **float get_size_y()**
  Retorna el alto de la imagen.
- **bool load()**
  Carga el mapa de bits de la ruta indicada, retorna verdadero en caso de éxito.
- **bool save()**
  Guarda el mapa de bits en la ruta actual, retorna verdadero en caso de éxito, si la ruta no ha sido cambiada se sobrescribirá el archivo.
- **bool destroy()**
  Elimina la imagen de la memoria, retorna verdadero en caso de éxito.
- **void set_target()**
  Asigna a la imagen como objetivo de las operaciones de dibujo.
- **bool lock()**
  Asegura la imagen en memoria para prepararlo contra múltiples accesos de lectura y escritura, retorna verdadero en caso de éxito.
- **void unlock()**
  Quita el bloqueo de la imagen liberando la memoria utilizada para los múltiples accesos.
- **ALLEGRO_COLOR get_pixel_color(Type_pos pos_x,Type_pos pos_y)**
  Retorna el color actual que tiene un pixel en la imagen pasando por parámetros su posición.
- **void draw()**
  Dibuja la imagen tomando en cuenta toda la configuración dada y las escalas globales para el tamaño real en cada eje.
- **void draw_in_another_target()**
  Dibuja la imagen sin tomar en cuenta las escalas globales, usado para dibujar en otro objetivo.

Operadores:

- **operator ALLEGRO_BITMAP*** ()
  Retorna un puntero **ALLEGRO_BITMAP** que representa al mapa de bits de la imagen para que pueda ser usado con las funciones restantes de Allegro.

Destructores:

- ∼ **Image**()
  Libera la memoria utilizada por la clase y elimina el mapa de bits cargado junto con todos los mapas de bits hijos generados.

2.10 Texto (LexRisLogic/Allegro5/Text.h)

2.10.1 Función: Iniciar Complemento Texto.

bool text_addon()
Inicia el complemento de texto y fuentes para su uso completo, retorna verdadero en caso de éxito.

2.10.2 Clase: Font

Clase que provee una interfaz para crear instancias de fuentes para el dibujado del texto en pantalla.
Funciones:

- **void set_path(std::string new_font_path)**
  Se asigna la ruta de la fuente a cargarse por la clase.
- **std::string get_path()**
  Retorna la ruta de la fuente a cargarse por la clase.
- **bool set_size(float new_size)**
  Asigna el tamaño de fuente, retorna verdadero si el tamaño es válido y aún no se haya cargado la fuente.
- **float get_size()**
  Retorna el tamaño de fuente.
- **bool load_ttf_font()**
  Carga la fuente de un archivo *ttf* tomando en cuenta la escala global del texto, retorna verdadero en caso de éxito.
- **bool load_ttf_font_for_another_target()**
  Carga la fuente de un archivo *ttf* sin tomar en cuenta la escala global del texto, retorna verdadero en caso de éxito.
- **bool destroy()**
  Destruye la fuente cargada actualmente, retorna verdadero en caso de éxito.

Operadores:

- **operator ALLEGRO_FONT* ()**
  Retorna un puntero **ALLEGRO_FONT** para que pueda ser usado con las funciones restantes de Allegro.

Destructores:

- **~ Font()**
  Libera la memoria utilizada por la clase y elimina la fuente cargada.

2.10.3 Clase: Text

Clase que dibuja una cadena de caracteres en pantalla.
Funciones:

- **void set_pos(Type_pos new_pos_x,Type_pos new_pos_y)**
  Cambia la posición del texto a (new_pos_x,new_pos_y).
- **void set_pos_x(Type_pos new_pos_x)**
  Cambia la posición del texto en el eje x.
- **Type_pos get_pos_x()**
  Retorna la posición actual del texto en el eje x.
- **void set_pos_y(Type_pos new_pos_y)**
  Cambia la posición del texto en el eje y.

- **Type_pos get_pos_y()**
  Retorna la posición actual del texto en el eje y.
- **void set_flag(int new_flag)**
  Cambia el modo en como se realizara la operación de dibujo del texto, puede tomar como valor los siguientes flags de Allegro:
  1. **ALLEGRO_ALIGN_LEFT**
     Dibuja el texto tomando la posición (x,y) como punto de partida, es decir el texto se extiende a la derecha (Igual que usar 0 como parámetro).
  2. **ALLEGRO_ALIGN_CENTRE** o **ALLEGRO_ALIGN_CENTER**
     Dibuja el texto tomando la posición (x,y) como punto central del texto.
  3. **ALLEGRO_ALIGN_RIGHT**
     Dibuja el texto tomando la posición (x,y) como punto final del dibujo, es decir el texto se extiende hacia la izquierda.
- **void set_color(ALLEGRO_COLOR new_color)**
  Cambia el color del texto.
- **ALLEGRO_COLOR get_color()**
  Retorna el color actual del texto.
- **void set_font(Font* new_font)**
  Cambia la fuente que se utilizará para la operación de dibujo del texto.
- **Font* get_font()**
  Retorna la fuente actual utilizada para la operación de dibujo del texto.
- **void draw()**
  Dibuja el texto con toda la configuración dada.

Operadores:

- **const char* operator = (const char* new_text)**
  Operador para asignar una cadena constante como texto a dibujar.
- **std::string operator = (std::string new_text)**
  Operador para asignar una cadena como texto a dibujar.
- **operator const char* ()**
  Retorna una cadena constante del texto que se dibuja en pantalla.
- **operator std::string ()**
  Retorna una cadena del texto que se dibuja en pantalla.

2.11 Mezclador (LexRisLogic/Allegro5/Mixer.h)

2.11.1 Función: Iniciar Complemento Audio.

bool audio_addon()
Inicia el complemento de audio para su uso completo, retorna verdadero en caso de éxito.

2.11.2 Función: Detener Complemento Audio.

void uninstall_audio()
Detiene el complemento de audio iniciado anteriormente, por defecto siempre se llama cuando termina la ejecución del programa.

2.11.3 Función: Restaurar Mezclador por defecto.

bool restore_default_mixer()
Restaura el mezclador y el dispositivo de salida de Allegro como mezclador por defecto para la reproducción de sonidos, retorna verdadero en caso de éxito; tener en cuenta que si se elimina un mezclador que se asigno como mezclador por defecto y aun se desea seguir usando la reproducción de sonidos, es necesario tener otro mezclador por defecto restaurando el mezclador de Allegro o creando un nuevo mezclador.

2.11.4 Clase: Mixer

Clase que representa al motor de sonido que usa Allegro para reproducir los sonidos de un archivo de audio o video conforme a la configuración de este.
Funciones:

- **bool set_frequency(unsigned int new_frequency)**
  Asigna la frecuencia con la que reproducirá la tarjeta de sonido, por defecto es 44100 Hz, retorna verdadero si el motor aun no ha sido iniciado para poder aplicar el cambio.
- **unsigned int get_frequency()**
  Retorna la frecuencia con la que reproducirá la tarjeta de sonido.
- **bool set_audio_depth(ALLEGRO_AUDIO_DEPTH new_frequency)**
  Asigna la profundidad, tipo y signo del sonido que se tomará en cuenta, por defecto usa 32 bits con precisión punto flotante, retorna verdadero si el motor aun no ha sido iniciado para poder aplicar el cambio, los valores que puede tomar son:
  1. **ALLEGRO_AUDIO_DEPTH_INT8**
  2. **ALLEGRO_AUDIO_DEPTH_INT16**
  3. **ALLEGRO_AUDIO_DEPTH_INT24**
  4. **ALLEGRO_AUDIO_DEPTH_FLOAT32**
  5. **ALLEGRO_AUDIO_DEPTH_UNSIGNED**
  6. **ALLEGRO_AUDIO_DEPTH_UINT8**
  7. **ALLEGRO_AUDIO_DEPTH_UINT16**
  8. **ALLEGRO_AUDIO_DEPTH_UINT24**
- **ALLEGRO_AUDIO_DEPTH get_audio_depth()**
  Retorna la macro de Allegro actualmente utilizada para definir la profundidad, tipo y signo de los sonidos.

- **bool set_channel_configuration(ALLEGRO_CHANNEL_CONF new_channel_conf)**
  Asigna la configuración de los altavoces, por defecto usa dos canales para sonido estéreo, retorna verdadero si el motor aun no ha sido iniciado para poder aplicar el cambio, los valores que puede tomar son:
  1. ALLEGRO_CHANNEL_CONF_1
  2. ALLEGRO_CHANNEL_CONF_2
  3. ALLEGRO_CHANNEL_CONF_3
  4. ALLEGRO_CHANNEL_CONF_4
  5. ALLEGRO_CHANNEL_CONF_5_1
  6. ALLEGRO_CHANNEL_CONF_6_1
  7. ALLEGRO_CHANNEL_CONF_7_1
- **ALLEGRO_CHANNEL_CONF get_channel_configuration()**
  Retorna la macro de Allegro actualmente usada para definir la configuración de los altavoces.
- **bool create()**
  Crea un mezclador con la configuración dada y una voz que representa un dispositivo de salida en el sistema con la misma configuración, excepto la profundidad que usa 16 bits con signo y precisión entera, retorna verdadero si se logro crear todo con éxito.
- **bool destroy()**
  Destruye el mezclador y la voz liberando la memoria utilizada.
- **bool set_quality(ALLEGRO_MIXER_QUALITY quality)**
  Asigna la calidad del mezclador de sonidos, retorna verdadero en caso de éxito, los valores que puede tomar son:
  1. ALLEGRO_MIXER_QUALITY_POINT
  2. ALLEGRO_MIXER_QUALITY_LINEAR
  3. ALLEGRO_MIXER_QUALITY_CUBIC
- **ALLEGRO_MIXER_QUALITY get_quality()**
  Retorna la calidad del mezclador de sonidos, por defecto el valor de la calidad es ALLEGRO_MIXER_QUALITY_LINEAR.
- **bool set_volume(float new_volume)**
  Asigna el volumen general del mezclador de sonidos donde 1.0 es el 100% pudiéndose duplicar el volumen al pasar 2.0 o disminuirlo con valores menores a 1.0, retorna verdadero en caso de éxito.
- **float get_volume()**
  Retorna el volumen general del mezclador de sonidos.
- **bool set_default_mixer()**
  Asigna al mezclador como mezclador por defecto para que las configuraciones realizadas se apliquen sobre la reproducción de sonido de archivos de audio y video.

Operadores:

- **operator ALLEGRO_MIXER* ()**
  Retorna un puntero ALLEGRO_MIXER para que pueda ser usado con las funciones restantes de Allegro.
- **operator ALLEGRO_VOICE* ()**
  Retorna un puntero ALLEGRO_VOICE para que pueda ser usado con las funciones restantes de Allegro.

Destructores:

- ~ **Mixer**()
  Elimina el mezclador y la voz, liberando la memoria utilizada por toda la clase.

2.12 Audio (LexRisLogic/Allegro5/Audio.h)

2.12.1 Clase: Audio

Clase que representa a un archivo de audio. **Funciones:**

- **void set_path(std::string new_audio_path)**
  Se asigna la ruta del archivo de audio que va a ser reproducido.
- **std::string get_path()**
  Retorna la ruta del archivo de audio que va a ser reproducido.
- **bool set_speed(float new_speed)**
  Asigna la velocidad de reproducción del audio, retorna verdadero en éxito.
- **float get_speed()**
  Retorna la velocidad de reproducción actual del audio.
- **bool set_pan(float new_pan)**
  Cambia la dirección de la reproducción del audio, los valores que toma la variable new_pan están entre [-1,1] donde:
  - Reproducción normal es 0
  - Reproducción por parlante izquierdo es -1
  - Reproducción por parlante derecho es 1
- **float get_pan()**
  Retorna la dirección de la reproducción del audio.
- **void set_volume(float new_volume)**
  Cambia el volumen de la reproducción del audio a escala del volumen asignado actualmente en el mezclador por defecto.
- **float get_volume()**
  Retorna el volumen asignado a la reproducción del audio.
- **void set_mode(ALLEGRO_PLAYMODE playmode)**
  Cambia el modo de reproducción del audio, retorna verdadero en caso de éxito, puede tomar los siguientes flags de Allegro:
  1. **ALLEGRO_PLAYMODE_ONCE**
     Reproduce el audio una sola vez.
  2. **ALLEGRO_PLAYMODE_LOOP**
     Reproduce el audio una y otra vez hasta que la reproducción sea detenida.
  3. **ALLEGRO_PLAYMODE_BIDIR**
     Reproduce el audio normalmente, pero una vez que llegue al final, la reproducción será de manera inversa hasta llegar al principio y reproducirse normalmente, esto seguirá hasta que la reproducción sea detenida.
- **ALLEGRO_PLAYMODE get_playmode()**
  Retorna el modo de reproducción del audio, por defecto el modo de reproducción es una sola vez.
- **bool load()**
  Carga el archivo de audio y le aplica toda la configuración dada, la configuración puede ser modificada a pesar de haber cargado ya el archivo.

- **bool destroy()**
  Elimina de memoria la carga del archivo de audio y su instancia para reproducción.
- **unsigned int size()**
  Retorna la posición final de reproducción del archivo de audio.
- **float get_time()**
  Retorna la duración del archivo de audio en segundos.
- **bool set_audio_position(unsigned int new_position)**
  Cambia la posición de la reproducción del audio, retorna verdadero en caso de éxito.
- **unsigned int get_audio_position()**
  Retorna la posición actual de la reproducción del audio.
- **bool is_playing()**
  Retorna verdadero si la reproducción esta activa.
- **void stop()**
  Detiene la reproducción.
- **void pause()**
  Pausa la reproducción.
- **void play()**
  Inicia o reanuda la reproducción.

Operadores:

- **operator ALLEGRO_SAMPLE* ()**
  Retorna un puntero **ALLEGRO_SAMPLE** para que pueda ser usado con las funciones restantes de Allegro.
- **operator ALLEGRO_SAMPLE_INSTANCE* ()**
  Retorna un puntero **ALLEGRO_SAMPLE_INSTANCE** para que pueda ser usado con las funciones restantes de Allegro.

Destructores:

- **~ Audio()**
  Libera la memoria utilizada por la clase y elimina el archivo de audio cargado.

2.13 Video (LexRisLogic/Allegro5/Video.h)

2.13.1 Función: Iniciar Complemento Video.

bool video_addon()
Inicia el complemento de video para su uso completo, retorna verdadero en caso de éxito.

2.13.2 Clase: Video

Clase que hereda de la clase base **BitmapBase** representando a un archivo de video.
Funciones:

- **void set_path(std::string new_video_path)**
  Se asigna la ruta del archivo de video a cargarse por la clase.
- **std::string get_path()**
  Retorna la ruta del archivo de video a cargarse por la clase.
- **float get_size_x()**
  Retorna el ancho del video.
- **float get_size_y()**
  Retorna el alto del video.
- **bool load()**
  Carga el video de la ruta indicada, retorna verdadero en caso de éxito.
- **bool destroy()**
  Elimina el video de la memoria, retorna verdadero en caso de éxito.
- **double get_video_position(ALLEGRO_VIDEO_POSITION_TYPE reference_position=ALLEGRO_VIDEO_POSITION_ACTUAL)**
  Retorna la posición actual en segundos de la reproducción del video, el parámetro es para tomar la posición en referencia:
  1. **ALLEGRO_VIDEO_POSITION_ACTUAL**
     A la posición en el sistema de Allegro.
  2. **ALLEGRO_VIDEO_POSITION_VIDEO_DECODE**
     A la posición a las imágenes del video.
  3. **ALLEGRO_VIDEO_POSITION_AUDIO_DECODE**
     A la posición del audio del video.
- **bool start()**
  Inicia la reproducción del video para poder obtener imágenes para dibujarlo en pantalla.
- **bool is_playing()**
  Retorna verdadero si la reproducción esta activa.
- **void stop()**
  Detiene la reproducción.
- **void pause()**
  Pausa la reproducción.
- **void play()**
  Reanuda la reproducción.
- **void draw()**
  Dibuja las imágenes del video tomando en cuenta toda la configuración dada y las escalas globales para el tamaño real en cada eje.

Operadores:

- **operator ALLEGRO_VIDEO*** ()
  Retorna un puntero **ALLEGRO_VIDEO** para que pueda ser usado con las funciones restantes de Allegro.

Destructores:

- ~ **Video**()
  Libera la memoria utilizada por la clase y elimina archivo de video cargado.

2.14 Iluminación (LexRisLogic/Allegro5/Lighting.h)

2.14.1 Clase: Lighting

Clase que representa a una iluminación básica usando un buffer que contiene intensidades de canal alfa en negro para crear zonas oscuras e iluminadas.
Constructores:

- **Lighting**()
  Obtiene las opciones de operación de dibujo como la operación de combinación, los valores de fuente y destino con el objetivo de retomar las verdaderas opciones de dibujo después de generar la iluminación.

Funciones:

- **void set_pos(Type_pos new_pos_x,Type_pos new_pos_y)**
  Cambia la posición del buffer de iluminación a (new_pos_x,new_pos_y).
- **void set_pos_x(Type_pos new_pos_x)**
  Cambia la posición del buffer de iluminación en el eje x.
- **Type_pos get_pos_x()**
  Retorna la posición actual del buffer de iluminación en el eje x.
- **void set_pos_y(Type_pos new_pos_y)**
  Cambia la posición del buffer de iluminación en el eje y.
- **Type_pos get_pos_y()**
  Retorna la posición actual del buffer de iluminación en el eje y.
- **float get_size_x()**
  Retorna el tamaño actual en el eje x del buffer de iluminación.
- **float get_size_y()**
  Retorna el tamaño actual en el eje y del buffer de iluminación.
- **bool create(int size_x,int size_y)**
  Crea un mapa de bits con el ancho y alto pasados por parámetros para ser el buffer de iluminación, retorna verdadero en caso de éxito.
- **bool destroy()**
  Elimina el buffer de iluminación, liberando la memoria utilizada, retorna verdadero en caso de éxito.
- **void set_target()**
  Asigna el buffer de iluminación como objetivo de las operaciones de dibujo, usar solo las funciones que brinda la clase para dibujar en el buffer con el objetivo de crear una correcta iluminación simple mediante figuras.
- **void clear(float intensity=0.0)**
  Limpia la buffer de iluminación con una intensidad de luz, 0 es totalmente opaco y 1 es totalmente iluminado.

- **template<typename T>**
  void add(T* object_shape,float intensity)
  Función que recibe una clase **T** que debe tener los siguientes métodos implementados:

  1. **ALLEGRO_COLOR get_color()** → Usado para obtener el color real del objeto y reponerlo terminada la llamada de la función.
  2. **void set_color(ALLEGRO_COLOR new_color)** → Usado para asignar la intensidad dada de iluminación mediante el uso de colores.
  3. **void draw_in_another_target()** → Usado para dibujar la al objeto en el buffer objetivo.

  La operación de dibujo consiste en dibujar al objeto con otro color que represente intensidad de iluminación, puede mandar figuras primitivas o crear figuras más complejas.
- **float get_pixel_intensity(Type_pos pos_x,Type_pos pos_y)**
  Retorna la intensidad de luz que tiene un pixel en el buffer de iluminación pasando por parámetros su posición.
- **void draw()**
  Aplica el buffer de iluminación en la imagen objetivo tomando en cuenta toda las escalas globales para el tamaño real en cada eje.
- **void draw_in_another_target()**
  Aplica el buffer de iluminación sin tomar en cuenta las escalas globales, usado para aplicar en otro objetivo.

Operadores:

- **operator ALLEGRO_BITMAP*** ()
  Retorna un puntero **ALLEGRO_BITMAP** que es el buffer de iluminación para que pueda ser usado con las funciones restantes de Allegro.

Destructores:

- ∼ **Lighting**()
  Destruye el buffer de iluminación y libera la memoria utilizada.

2.15 Interfaz (LexRisLogic/Allegro5/Special/Interface.h)

2.15.1 Clase Base: Interface

Clase Base que contiene la estructura base de una interfaz para manejar entradas del usuario en pantalla, clase especial que depende de las clases de **LL_AL5**.

Constructores:

- **Interface**()
  Constructor para los valores constantes usados por las clases que heredan de él.

Funciones:

- **void set_pos(Type_pos new_pos_x,Type_pos new_pos_y)**
  Cambia la posición de la interfaz a (new_pos_x,new_pos_y).
- **void set_pos_x(Type_pos new_pos_x)**
  Cambia la posición de la interfaz en el eje x.
- **Type_pos get_pos_x()**
  Retorna la posición actual de la interfaz en el eje x.
- **void set_pos_y(Type_pos new_pos_y)**
  Cambia la posición de la interfaz en el eje y.
- **Type_pos get_pos_y()**
  Retorna la posición actual de la interfaz en el eje y.
- **float get_size_x()**
  Retorna el ancho de la interfaz.
- **float get_size_y()**
  Retorna el alto de la interfaz.
- **void set_unclick_line_color(ALLEGRO_COLOR new_unclick_line_color)**
  Cambia el color del borde de la interfaz cuando no esta enfocada.
- **ALLEGRO_COLOR get_unclick_line_color()**
  Retorna el color del borde de la interfaz cuando no esta enfocada.
- **void set_unclick_fill_color(ALLEGRO_COLOR new_unclick_fill_color)**
  Cambia el color del relleno de la interfaz cuando no esta enfocada.
- **ALLEGRO_COLOR get_unclick_fill_color()**
  Retorna el color del relleno de la interfaz cuando no esta enfocada.
- **void set_click_line_color(ALLEGRO_COLOR new_unclick_line_color)**
  Cambia el color del borde de la interfaz cuando esta enfocada.
- **ALLEGRO_COLOR get_click_line_color()**
  Retorna el color del borde de la interfaz cuando esta enfocada.
- **void set_click_fill_color(ALLEGRO_COLOR new_unclick_fill_color)**
  Cambia el color del relleno de la interfaz cuando esta enfocada.
- **ALLEGRO_COLOR get_click_fill_color()**
  Retorna el color del relleno de la interfaz cuando esta enfocada.
- **void set_thickness(float new_thickness)**
  Cambia el grosor del borde de la interfaz.
- **float get_thickness()**
  Retorna el grosor del borde de la interfaz.
- **Font* get_font()**
  Retorna la fuente utilizada para el texto de la interfaz.
- **void set_text_color(ALLEGRO_COLOR new_text_color)**
  Cambia el color del texto de la interfaz.
- **ALLEGRO_COLOR get_text_color()**
  Retorna el color del texto de la interfaz.

2.15.2 Clase: Button

Clase que hereda de la clase base **Interface** representando a un botón, clase especial que depende de las clases de **LL_AL5**.
Constructores:

- **Button(Input* input_pointer)**
  Indica cual va a ser la interfaz de entrada a usarse para obtener los eventos de selección sobre el botón.

Funciones:

- **void set_button_text(std::string new_button_text)**
  Cambia el texto del botón.
- **std::string get_button_text()**
  Retorna el texto del botón.
- **void set_font(Font* new_font)**
  Cambia la fuente utilizada para dibujar el texto del botón.
- **void draw()**
  Dibuja el botón en la pantalla objetivo tomando en cuenta las escalas globales, a su vez lee las entradas por si el usuario genero el evento de selección del botón.
- **bool is_clicked()**
  Retorna el estado del botón, verdadero si ha sido seleccionado, tener en cuenta que el evento solo será transmitido una sola vez tornándose falso en su lectura.
- **bool button_on()**
  Activa la generación del evento de selección del botón.
- **bool button_off()**
  Desactiva la generación del evento de selección del botón.

2.15.3 Clase: TextBox

Clase que hereda de la clase base **Interface** representando a una caja de texto, clase especial que depende de las clases de **LL_AL5**.
Constructores:

- **TextBox(Input* input_pointer)**
  Indica cual va a ser la interfaz de entrada a usarse para obtener los eventos de selección sobre la caja de texto.

Funciones:

- **void set_hide_option(bool hide_option)**
  Cambia la opción para ocultar el texto ingresado por el carácter *'*'*.
- **bool get_hide_option()**
  Retorna verdadero si la opción de texto oculto esta activada.
- **void set_text_length(unsigned int new_text_length)**
  Cambia el máximo tamaño que puede recibirse de entrada del usuario.
- **unsigned int get_text_length()**
  Retorna el máximo tamaño que puede recibirse de entrada del usuario.
- **void set_font(Font* new_font)**
  Cambia la fuente utilizada para dibujar el texto de la interfaz.
- **void clear()**
  Limpia la entrada vaciando la cadena de texto.
- **bool set_value(std::string new_value)**
  Cambia el valor actual en la caja de texto, retorna verdadero en caso de éxito.

- **std::string get_value()**
  Retorna la cadena que contiene la caja de texto.
- **void draw()**
  Dibuja la caja de texto en la pantalla objetivo tomando en cuenta las escalas globales, a su vez lee las entradas por si el usuario genero el evento de selección de la caja de texto y también cuando es enfocada se activa la entrada de una cadena en la clase **Input** para obtener la entrada del usuario en la cadena de la caja de texto.
- **bool is_clicked()**
  Retorna el estado de la caja de texto, verdadero si esta enfocada.
- **bool textbox_on()**
  Activa la generación del evento de selección de la caja de texto, así como la opción de ingresado de datos por la clase **Input**.
- **bool textbox_off()**
  Desactiva la generación de eventos y desactiva la entrada de la cadena mediante eventos de la clase **Input**.

Destructores:

- ∼ **TextBox()**
  Desactiva la entrada de la cadena mediante eventos de la clase **Input** por si el programador olvido de desactivar la entrada y libera la memoria utilizada.

2.16 Sprite (LexRisLogic/Allegro5/Special/Sprite.h)

2.16.1 Clase: Sprite

Clase que hereda de la clase base **BitmapBase** representando a un conjunto de imágenes, clase especial que depende de las clases de **LL_AL5**.
Funciones:

- **float get_size_x()**
  Retorna el ancho del sprite actualmente seleccionado.
- **float get_size_y()**
  Retorna el alto del sprite actualmente seleccionado.
- **unsigned int get_size()**
  Retorna el número de sprites que existen almacenados.
- **bool set_size(unsigned int new_size)**
  Cambia el número de sprites que se almacenaran, retorna verdadero si aún no se ha separado memoria para la creación de los sprites y si el tamaño es válido.
- **bool set_selection(unsigned int new_selection)**
  Cambia el sprite seleccionado para las operaciones de dibujo, obtención de su tamaño y usarlo como objetivo otras operaciones de dibujo, retorna verdadero si el cambio es válido.
- **unsigned int get_selection()**
  Retorna el índice del sprite seleccionado, el índice toma valores en el intervalo de [0:size>.
- **bool create()**
  Separa la memoria para poder crear los mapas de bits necesarios para todos los sprites, creando también arreglos para obtener el tamaño de cada uno de estos, retorna verdadero en caso de éxito.
- **bool create_selection(int size_x,int size_y)**
  Crea un mapa de bits vacío de tamaño *(size_x,size_y)* en la posición seleccionada, si ya existe un mapa de bits este será eliminado para crear el nuevo mapa de bits, retorna verdadero en caso de éxito.
- **bool destroy_selection()**
  Elimina el mapa de bits en la posición actualmente seleccionada, retorna verdadero en caso de éxito.
- **bool create_data_from_directory(std::string sprites_path,std::string bitmap_format)**
  Crea todos los mapas de bits cargando los sprites del disco los cuales deberán tener su ruta completa de la siguiente manera:
  sprites_path (index_number+1) bitmap_format
  Por ejemplo se cargaran dos sprites que tienen la siguiente ruta, ***/home/user/bitmaps/sprite*** y en formato ***.bmp***, entonces los dos sprites deberán tener como ruta completa:
  /home/user/bitmaps/sprite1.bmp
  /home/user/bitmaps/sprite2.bmp
- **bool destroy_data()**
  Elimina todos los mapas de bits creados o cargados, retorna verdadero en caso de éxito.
- **bool destroy()**
  Elimina y libera la memoria separada para la creación y utilización de los mapas de bits.
- **void set_target()**
  Asigna al mapa de bits actualmente seleccionado como objetivo de las operaciones de dibujo.

- **bool lock()**
  Asegura el mapa de bits actualmente seleccionado en memoria para prepararlo contra múltiples accesos de lectura y escritura, retorna verdadero en caso de éxito.
- **void unlock()**
  Quita el bloqueo del mapa de bits actualmente seleccionado liberando la memoria utilizada para los múltiples accesos.
- **ALLEGRO_COLOR get_pixel_color(Type_pos pos_x,Type_pos pos_y)**
  Retorna el color actual que tiene un pixel en el mapa de bits actualmente seleccionado pasando por parámetros su posición.
- **void draw()**
  Dibuja el mapa de bits actualmente seleccionado tomando en cuenta toda la configuración dada y las escalas globales para el tamaño real en cada eje.
- **void draw_in_another_target()**
  Dibuja el mapa de bits actualmente seleccionado sin tomar en cuenta las escalas globales, usado para dibujar en otro objetivo.

Operadores:

- **operator ALLEGRO_BITMAP* ()**
  Retorna un puntero **ALLEGRO_BITMAP** que representa al mapa de bits actualmente seleccionado para que pueda ser usado con las funciones restantes de Allegro.

Destructores:

- **~ Sprite()**
  Elimina todos los mapas de bits creados y libera toda la memoria utilizada por la clase.

3 namespace LL_ENet

Espacio de nombres que incluye código amigable para manejar la capa de comunicación red que nos brinda la librería ENet.

3.1 ENet (LexRisLogic/ENet/ENet.h)

3.1.1 Función: Instalar Sistema ENet.

bool install_enet()
Instala el sistema ENet para poder ser utilizado por el software, retorna verdadero en caso de éxito.

3.1.2 Función: Desinstalar Sistema ENet.

void uninstall_enet()
Desinstala el sistema ENet cerrando conexiones del software con el sistema.

3.2 Servidor (LexRisLogic/ENet/Server.h)

3.2.1 Clase: Server

Clase que representa a un servidor con conexiones peer-to-peer con protocolo UDP.
Funciones:

- **bool set_ip(std::string new_ip)**
  Cambia la dirección IP a la que pertenece el servidor, retorna falso si la dirección IP no es válida o si el servidor ya esta corriendo.
- **std::string get_ip()**
  Retorna la dirección IP asignada para el servidor.
- **bool set_port(unsigned int new_port)**
  Cambia el puerto asignado al servidor, retorna falso si el servidor ya esta corriendo.
- **unsigned int get_port()**
  Retorna el puerto asignado al servidor.
- **bool set_max_peers_connections(unsigned int max_peers_connections)**
  Cambia el número de conexiones máximas permitidas por el servidor, retorna falso si el servidor ya esta corriendo.
- **unsigned int get_max_peers_connections()**
  Retorna el número de conexiones máximas permitidas por el servidor.
- **void set_wait_time(unsigned int wait_time)**
  Cambia el tiempo de espera en milisegundos para los eventos generados por el servidor.
- **unsigned int get_wait_time()**
  Retorna el tiempo de espera para los eventos.
- **bool start_server()**
  Inicia el servidor para recibir conexiones y enviar mensajes, retorna verdadero si se ha logrado crear un Host con la dirección IP y el puerto asignado.
- **bool get_event()**
  Revisa la cola de eventos del servidor, retorna verdadero si se encuentra un evento en la cola de eventos.
- **bool get_new_connection()**
  Retorna verdadero si el evento generado es una nueva conexión de un cliente.

- **bool get_disconnection()**
  Retorna verdadero si el evento generado es una desconexión de un cliente.
- **ENetPeer* get_peer_connected()**
  Retorna un puntero a una conexión peer de un cliente que se ha conectado.
- **ENetPeer* get_peer_disconnected()**
  Retorna un puntero a una conexión peer de un cliente que se ha desconectado.
- **ENetPeer* get_issuer_peer()**
  Retorna un puntero a una conexión peer de un cliente propietario del actual mensaje en la cabeza de la cola de paquetes.
- **std::string get_message_received()**
  Retorna el actual mensaje en la cabeza de la cola de paquetes.
- **bool remove_message_received()**
  Elimina el mensaje en la cabeza de la cola de paquetes, retorna verdadero si el mensaje fue removido.
- **void clear()**
  Vacía la cola de paquetes.
- **bool empty()**
  Retorna verdadero si la cola de paquetes esta vacía.
- **bool send(ENetPeer* receiver, std::string message)**
  Envía un mensaje a una conexión peer de un cliente, retorna verdadero si se envió por la red con éxito.
- **bool broadcast(std::string message)**
  Envía un mensaje a todos los clientes conectados al servidor, retorna verdadero si se envió por la red con éxito.
- **bool stop_server()**
  Detiene el servidor, limpiando la cola de paquetes y cerrando todas las conexiones.

Destructores:

- **~ Server()**
  Detiene el servidor y libera toda la memoria utilizada por la clase.

3.3 Cliente (LexRisLogic/ENet/Client.h)

3.3.1 Clase: Client

Clase que representa a un cliente con conexiones peer-to-peer con protocolo UDP.
Funciones:

- **bool set_ip(std::string new_ip)**
  Establece la dirección IP del servidor al cual se conectara el cliente.
- **std::string get_ip()**
  Retorna la dirección IP del servidor al cual se conectara el cliente.
- **bool set_port(unsigned int new_port)**
  Establece el puerto del servidor al cual se conectara el cliente.
- **unsigned int get_port()**
  Retorna el puerto del servidor al cual se conectara el cliente.
- **bool set_test_time(unsigned int new_test_time)**
  Cambia el tiempo de prueba en milisegundos para obtener la conexión al servidor, retorna falso si el tiempo es 0 o si ya se inicio el Host propio del cliente.

- **unsigned int get_test_time()**
  Retorna el tiempo de prueba de conexión.
- **void set_wait_time(unsigned int wait_time)**
  Cambia el tiempo de espera en milisegundos para los eventos generados por el cliente.
- **unsigned int get_wait_time()**
  Retorna el tiempo de espera para los eventos.
- **bool start_client()**
  Inicia el Host del cliente preparándolo para una futura conexión con un servidor, retorna verdadero si se ha logrado crear con éxito el Host del cliente.
- **bool connect_to_server()**
  Inicia la conexión con el servidor, si lo consigue probará la conexión para informar al servidor que se ha conectado un cliente, retorna verdadero en caso de éxito.
- **bool disconnect_from_server()**
  Vacía la lista de eventos del cliente y se desconecta del servidor cuando recibe la señal de desconexión por parte del servidor.
- **bool get_connection_status()**
  Retorna el estado de la conexión del cliente con el servidor.
- **bool get_event()**
  Actualiza todas las variables del servidor y retorna verdadero si se genero algún evento en un tiempo definido anteriormente.
- **std::string get_message_received()**
  Retorna el actual mensaje recibido por el servidor en la cola de paquetes.
- **bool remove_message_received()**
  Elimina el mensaje en la cabeza de la cola de paquetes, retorna verdadero si el mensaje fue removido.
- **void clear()**
  Vacía la cola de paquetes.
- **bool empty()**
  Retorna verdadero si la cola de paquete esta vacía.
- **bool send(std::string message)**
  Envía un mensaje al servidor, retorna verdadero si se envió por la red con éxito.
- **bool stop_client()**
  Detiene la conexión con el servidor solo si se encuentra aun conectado, detiene el host del cliente y limpia la cola de paquetes.

Destructores:

- **~ Client()**
  Detiene el Host del cliente y libera toda la memoria utilizada por la clase.

4 namespace LL_irrKlang

Espacio de nombres que incluye un código amigable para manejar el motor de sonido irrKlang, usando la API de irrKlang.

4.1 Motor de Sonido (LexRisLogic/irrKlang/irrKlang.h)

4.1.1 Clase: SoundEngine

Clase que representa al motor de sonido irrKlang.
Funciones:

- **bool create()**
  Crea un motor de sonido irrKlang para poder reproducir archivos de audio.
- **void set_default_engine()**
  Establece al objeto como motor de sonido por defecto.
- **void set_volume(float new_volume)**
  Establece el volumen principal del motor donde el valor de 1.0 es normal.
- **float get_volume()**
  Obtiene el volumen principal actual del motor.
- **bool destroy()**
  Destruye el motor de sonido irrKlang del objeto.

Operadores:

- **operator irrklang::ISoundEngine* ()**
  Retorna un puntero al motor de sonido irrKlang para manejarlo con las demás funciones de irrKlang.

Destructores:

- **~ SoundEngine()**
  Destruye el motor de sonido irrKlang del objeto y libera la memoria utilizada por el objeto.

4.1.2 Variable: extern SoundEngine* default_engine

Puntero al principal motor de sonido irrKlang, que es usado para poder reproducir archivos de audio.

4.2 Audio (LexRisLogic/irrKlang/Audio.h)

4.2.1 Clase: Audio

Funciones:

- **void set_path(std::string new_audio_path)**
  Se asigna la ruta del archivo de audio que va a ser reproducido por el motor de sonido.
- **std::string get_path()**
  Retorna la ruta del archivo de audio que va a ser reproducido por el motor de sonido.
- **bool set_speed(float new_speed)**
  Asigna la velocidad de reproducción del audio, retorna verdadero en éxito.
- **float get_speed()**
  Retorna la velocidad de reproducción actual del audio.

- **bool set_pan(float new_pan)**
  Cambia la dirección de la reproducción del audio. Los valores que toma la variable new_pan están entre [-1,1] donde:
  - Reproducción normal es 0
  - Reproducción por parlante izquierdo es -1
  - Reproducción por parlante derecho es 1
- **float get_pan()**
  Retorna la dirección de la reproducción del audio.
- **void set_volume(float new_volume)**
  Cambia el volumen de la reproducción del audio a escala del volumen asignado actualmente en el motor de sonido irrKlang.
- **float get_volume()**
  Retorna el volumen asignado a la reproducción del audio.
- **bool set_loop_mode(bool loop_mode_on)**
  Cambia el modo de reproducción por ciclos.
- **bool get_loop_mode()**
  Retorna verdadero si se el modo de la reproducción es por ciclos.
- **bool load()**
  Carga el archivo de audio como audio irrKlang para poder reproducirlo, retorna verdadero en caso de éxito.
- **unsigned int size()**
  Retorna la duración en milisegundos del archivo de audio.
- **bool set_position(unsigned int new_position)**
  Establece la posición de reproducción enviada por parámetro en milisegundos, retorna verdadero en caso de éxito.
- **unsigned int get_position()**
  Retorna la posición actual de la reproducción del audio.
- **bool is_playing()**
  Retorna verdadero si la reproducción esta activa.
- **void stop()**
  Detiene la reproducción.
- **void pause()**
  Pausa la reproducción.
- **void play()**
  Inicia o reanuda la reproducción.
- **bool destroy()**
  Destruye el audio irrKlang creado para la reproducción del archivo de audio.

Operadores:

- **operator irrklang::ISound* ()**
  Retorna un puntero al audio irrKlang para manejarlo con las demás funciones de irrKlang.

Destructores:

- **~ Audio()**
  Destruye el audio irrKlang y libera la memoria utilizada por el objeto.

5 namespace LL_MathStructure

Espacio de nombres que incluye un conjunto de estructuras matemáticas y funciones geométricas para su interacción.

5.1 Punto (LexRisLogic/MathStructures/Point.h)

5.1.1 Clase: Point

Clase que representa a un punto en una dimensión.
Constructores:

- **Point()**
  Crea el punto sin definir la dimensión.
- **Point(unsigned int dimension)**
  Crea el punto definiendo la dimensión en que trabajara y dando como valor 0 en cada eje.
- **Point(const Point& another_point)**
  Constructor para copia profunda, usado para hacer una copia real de la estructura interna de la clase sin alterar la información.

Funciones:

- **bool set_dimension(unsigned int new_dimension)**
  Cambia la dimensión del objeto, eliminando el anterior punto y creando otro con valor 0 en cada eje dimensional.
- **unsigned int get_dimension() const**
  Retorna el valor de la dimensión del punto.

Operadores:

- **float& operator [] (unsigned int index)**
  Retorna el eje dimensional indicado por el índice, por referencia para poder ser modificado.
- **const float operator [] (unsigned int index) const**
  Retorna el eje dimensional indicado por el índice, el objetivo de esta réplica es dar un error si se trata de cambiar el valor en una dimensión de un punto constante.
- **Point& operator = (const Point& another_point)**
  Copia la dimensión y todos los valores de cada eje dimensional de otro punto.
- **bool operator == (const Point& another_point) const**
  Retorna verdadero si el otro punto es igual al que invoca la función.
- **bool operator != (const Point& another_point) const**
  Retorna verdadero si el otro punto es diferente al que invoca la función.

Destructores:

- **~ Point()**
  Elimina al punto y libera la memoria utilizada.

5.1.2 Operador: << para la clase Point

std::ostream& operator << (std::ostream& output_stream,Point point)
Operador de salida estándar en consola o terminal del punto.

5.1.3 Función: Distancia Euclidiana

double euclidean_distance(Point first_point,Point second_point)
Retorna la distancia euclidiana entre dos puntos si pertenecen a la misma dimensión.

5.1.4 Función: Crear punto 2D

Point create_point(float x,float y)
Crea un punto de dimensión dos.

5.1.5 Función: Crear punto 3D

Point create_point(float x,float y,float z)
Crea un punto de dimensión tres.

5.2 Segmento (LexRisLogic/MathStructures/LineSegment.h)

5.2.1 Clase: LineSegment

Clase que representa a un segmento de recta en cualquier dimensión.
Constructores:

- **LineSegment()**
  Crea un segmento sin definir la dimensión.
- **LineSegment(unsigned int dimension)**
  Crea un segmento donde el punto de inicio y final tienen el valor del punto de origen en la dimensión asignada, cuando un segmento tiene los dos puntos iguales hacen que no sea válido para las funciones de intersección.

Funciones:

- **bool set_dimension(unsigned int new_dimension)**
  Cambia la dimensión del segmento asignando al punto de inicio y final el valor del punto de origen.
- **unsigned int get_dimension()**
  Retorna el valor de la dimensión del segmento.
- **bool set_ini_point(Point new_ini_point)**
  Cambia el punto inicial del segmento solo si pertenece a la misma dimensión, retorna verdadero en caso de éxito.
- **const Point get_ini_point()**
  Retorna el punto inicial del segmento.
- **bool set_end_point(Point new_end_point)**
  Cambia el punto final del segmento solo si pertenece a la misma dimensión, retorna verdadero en caso de éxito.
- **const Point get_end_point()**
  Retorna el punto final del segmento.
- **bool in_range(unsigned int dimension,float number)**
  Retorna verdadero si un número se encuentra en el rango dimensional seleccionado del segmento.

5.2.2 Función: Intersección de Lineas en Dos Dimensiones

bool intersection_of_lines_in_two_dimensions(LineSegment first_line,
LineSegment second_line,
float* x=nullptr,float* y=nullptr)
Retorna la intersección entre dos rectas de dos dimensiones, retorna tres valores, Si existe intersección entre dos rectas y el punto de intersección; puede evitar mandar las variables x e y, para no obtener el valor de intersección; la función retorna falso si se envían rectas paralelas o incorrectas.

5.2.3 Función: Intersección de Segmentos en Dos Dimensiones

bool intersection_of_line_segments_in_two_dimensions(LineSegment first_segment, LineSegment second_segment, float* x=nullptr,float* y=nullptr)

Retorna la intersección entre dos rectas de dos dimensiones, retorna tres valores, Si existe intersección entre dos segmentos y el punto de intersección; puede evitar mandar las variables x e y, para no obtener el valor de intersección; la función retorna falso si se envían segmentos paralelos o incorrectos; los valores de x e y pueden haber sido modificados incluso aunque no exista intersección entre los segmentos.

5.3 Polígono (LexRisLogic/MathStructures/Polygon.h)

Clase que representa un polígono de puntos de dos dimensiones.

5.3.1 Clase: Polygon

Funciones:

- **bool add_point(Point point)**
  Añade al polígono un punto de segunda dimensión al final de la lista, retorna verdadero en caso de éxito.
- **bool remove_point(unsigned int index)**
  Remueve el punto del polígono de la posición elegida, retorna verdadero en caso de éxito.
- **unsigned int size()**
  Retorna el número de puntos que tiene el polígono.
- **void clear()**
  Vacía la lista de puntos del polígono.
- **bool set_point(unsigned int Point,Point new_point)**
  Cambia el punto del polígono en la posición indicada si el nuevo punto pertenece a la segunda dimensión, retorna verdadero en caso de éxito.

Operadores:

- **const Point operator [] (unsigned int index)**
  Retorna el punto del polígono en la posición indicada.

5.3.2 Función: Punto dentro de un Polígono

bool point_into_polygon(Polygon polygon,Point point)
Retorna verdadero si el punto de segunda dimensión enviado por parámetro se encuentra en el interior del polígono.

5.3.3 Función: Colisión de Polígonos

bool collision_of_polygons(Polygon first_polygon,Polygon second_polygon, std::list<Point>* points=nullptr)
Retorna dos valores, verdadero o falso si existe colisión entre dos polígonos enviados y un puntero a una lista donde se guardaran todos los puntos de intersección.

6 namespace LL_DataStructure

Espacio de nombres que incluye estructuras de datos y funciones para para su interacción.

6.1 Matriz Dispersa (LexRisLogic/DataStructures/SparseMatrix.h)

6.1.1 Clase: SparseMatrix

template<typename T>
Matriz Dispersa que almacena los datos diferentes al valor nulo, es decir, valor que representa al vacío en la matriz con el objetivo de disminuir la memoria utilizada.
Constructores:

- **SparseMatrix(unsigned int size_x,unsigned int size_y,T null_value)**
  Crea una Matriz de dimensión [size_x,size_y] vacía y se pasa el valor que representara el vacío en la matriz.

Clases:

- **Class_Controller**
  Clase controladora de una posición de la matriz, se encarga de asignarle valor o eliminar el valor en dicha posición actualizando la matriz dispersa.
  Constructores:
  - **Class_Controller(__SparseMatrixNodeBase__** pointer_x, __SparseMatrixNodeBase__** pointer_y, unsigned int pos_x,unsigned int pos_y,T null_value)**
    Solo puede ser usado por la matriz dispersa de manera correcta, por lo que ya existe una función que nos retorna esta clase ya configurada, no existen controladores vacíos.

  Funciones:
  - **T get_value()**
    Retorna una copia del dato que contiene en la matriz en tal posición.

  Operadores:
  - **T operator = (T new_data)**
    Operador para asignar el dato que guardara la matriz en la posición requerida, si el dato es el valor nulo se eliminará de memoria el objeto en esa posición.
  - **operator T ()**
    Operador de conversión automática para evitar llamar a get_value() y tener un código más legible.

Funciones:

- **unsigned int get_size_x()**
  Retorna el tamaño de la matriz en el eje x.
- **unsigned int get_size_y()**
  Retorna el tamaño de la matriz en el eje y.
- **T get_null_value()**
  Retorna el valor que representa al vacío en la matriz dispersa.
- **void clear()**
  Elimina todos los datos almacenados en la matriz dispersa y libera la memoria utilizada por estos.

Operadores:

- **Class_Controller operator** () (**unsigned int pos_x,unsigned int pos_y**)
  Retorna el controlador de la matriz dispersa en para posición pasada por parámetro.

Destructores:

- ∼ **SparseMatrix**()
  Destruye todos los objetos en la matriz y libera la memoria utilizada por la estructura.

6.2 Árbol-R (LexRisLogic/DataStructures/RTree.h)

6.2.1 Estructura: MBB

Estructura que contiene dos puntos en una dimensión para representar el cuadro mínimo delimitador de un objeto en el espacio.
Constructor:

- **MBB**()
  Crea los puntos representantes del cuadro mínimo delimitador sin asignarles una dimensión.
- **MBB**(**unsigned int new_dimension**)
  Inicia los puntos representantes del cuadro mínimo delimitador pasando la dimensión a la que pertenece.

Variables:

- **unsigned int dimension=0**
  Dimensión del cuadro mínimo delimitador.
- **LL_MathStructure::Point first_point**
  Primer Punto representante del cuadro mínimo delimitador.
- **LL_MathStructure::Point second_point**
  Segundo Punto representante del cuadro mínimo delimitador.

Funciones:

- **bool set_dimension**(**unsigned int new_dimension**)
  Actualiza la dimensión del cuadro mínimo delimitador y los puntos representantes.

Operadores:

- **bool operator** == (**MBB another_mbb**)
  Compara los dos puntos representantes de otro cuadro mínimo delimitador, retorna verdadero si ambos puntos son iguales.
- **bool operator** != (**MBB another_mbb**)
  Compara los dos puntos representantes de otro cuadro mínimo delimitador, retorna verdadero si algún punto es diferente.

6.2.2 Función: Distancia entre Cuadros Mínimos Delimitadores

double mbb_distance(**MBB first_mbb,MBB second_mbb**)
Retorna la distancia mínima entre dos cuadros mínimos delimitadores.

6.2.3 Clase: RTree

template<typename T,unsigned int DIMENSION,unsigned int NODE_SIZE>
Estructura de Datos Espacial para guardar objetos de acuerdo a una función que se pasa por parámetro para hallar el Cuadro Mínimo Delimitador de estos y así poder ubicarlos en el espacio, y también se pasa el tamaño de los nodos para evitar la sobreposición.
Constructores:

- **RTree(MBB (*Function_to_mbb)(T))**
  Recibe un puntero a una función que reciba el tipo de dato que guardará la estructura y devuelva su respectivo Cuadro Mínimo Delimitador, la estructura usara esta función para encontrar su posición en el espacio e insertarlos en la estructura, tener en cuenta que la función debe ser biyectiva porque es usada para las operaciones de búsqueda y eliminado.

Clases:

- **iterator**
  Iterador para recorrer por todos los elementos de la estructura, retorna una copia del objeto (solo lectura).
  Constructores:
  - **iterator(__RTreeNodeBase__* root_node=nullptr)**
    Recibe un puntero al nodo del Árbol-R, este nodo sera tratado como el nodo raíz y generara la estructura del iterador para poder recorrer los datos del nodo; por defecto se pasa un puntero nulo para crear iteradores vacíos.

  Operadores:
  - **iterator operator ++ (int)**
    Crea una copia del iterador actual, luego itera al siguiente elemento actualizando la estructura y retorna la copia.
  - **iterator operator ++ ()**
    Itera al siguiente elemento y actualiza la estructura sin crear una copia, se retorna a si mismo con el cambio realizado.
  - **const T operator * ()**
    Accede a una copia del dato que actualmente guarda el iterador.
  - **bool operator != (iterator another_iterator)**
    Retorna verdadero si dos iteradores son diferentes, es decir, que no se refieran al mismo objeto.
- **Class_NodeIterator**
  Iterador para recorrer el árbol a través de sus cuadros mínimos delimitadores y acceder hasta los datos con el objetivo de visualización del árbol.
  Constructores:
  - **Class_NodeIterator(__RTreeNodeBase__* node=nullptr)**
    Recibe un puntero al nodo del Árbol-R, para poder obtener los datos del nodo; por defecto se pasa un puntero nulo para crear iteradores de nodos vacíos.

  Funciones:
  - **unsigned int size()**
    Retorna el número de datos que guarda el nodo, si es un nodo padre los datos son nodos hijos.
  - **bool get_type()**
    Retorna verdadero si el nodo es de tipo padre, falso si es de tipo hijo.

- **MBB get_mbb()**
  Retorna el cuadro mínimo delimitador del nodo.
- **T get_data(unsigned int index)**
  Retorna el dato que se guarda en la posición enviada, solo los nodos hijos almacenan estos datos.
- **Class_NodeIterator get_son(unsigned int index)**
  Retorna un iterador de nodo que se guarda en la posición enviada, solo los nodos padres almacenan estos nodos hijos.
- **Class_NodeIterator get_parent()**
  Retorna el nodo padre en un iterador de nodo, es necesario verificar que el iterador de nodo sea válido.
- **bool is_valid()**
  Retorna verdadero si el iterador de nodo puede acceder a la información del nodo que guarda.

Funciones:

- **iterator begin()**
  Retorna un iterador al inicio de la estructura.
- **iterator end()**
  Retorna un iterador vacío que representa el final de la estructura.
- **Class_NodeIterator get_node_iterator()**
  Retorna un iterador de nodo que guarda la información de la raíz del Árbol-R.
- **unsigned int size()**
  Retorna el número de elementos que actualmente guarda la estructura.
- **void clear()**
  Elimina todos los datos almacenados en la estructura.
- **bool find(T data)**
  Busca si un dato ya ha sido insertado.
- **bool insert(T new_data)**
  Inserta un nuevo elemento a la estructura, retorna falso si el cuadro mínimo delimitador no pertenece a la dimensión o si ya se encuentra un dato con igual cuadro mínimo delimitador.
- **bool remove(T data_to_remove)**
  Remueve un dato de la estructura, retorna verdadero si se encuentra y se logra eliminar.
- **std::list<T> range_query(MBB mbb)**
  Retorna todos los elementos que se encuentren dentro del Cuadro Mínimo Delimitador pasado por parámetro.

Destructores:

- **∼ RTree()**
  Elimina todos los datos de la estructura y libera toda la memoria utilizada por la estructura.

7 namespace LL

Espacio de Nombres que incluye funciones y clases básicas listadas más adelantes.

7.1 Conversión de Tipo (LexRisLogic/Convert.h)

7.1.1 Función: Conversión de Cualquier Tipo a Cadena.

template<typename T>
std::string to_string(T data)
Convierte el objeto a una cadena, para poder convertir el objeto es necesario tener sobrecargado el operador <<(**ostream**, **T**).

7.1.2 Función: Conversión de Cadena a Entero.

int to_int(std::string data)
Convierte la cadena a un entero, la cadena solo debe contener los caracteres numéricos para poder ser convertida.

7.1.3 Función: Conversión de Cadena a Punto Flotante Simple.

float to_float(std::string data)
Convierte la cadena a un punto flotante simple, la cadena solo debe tener caracteres numéricos y un '.' separando la parte entera de la parte flotante.

7.1.4 Función: Conversión de Cadena a Punto Flotante Doble.

double to_double(std::string data)
Convierte la cadena a un punto flotante doble, la cadena solo debe tener caracteres numéricos y un '.' separando la parte entera de la parte flotante.

7.2 Encriptador (LexRisLogic/Encryptor.h)

7.2.1 Constante: const char* const DEFAULT_DICTIONARY

Diccionario de caracteres por defecto que usará el encriptador de cadenas.

7.2.2 Clase: Encryptor

Clase que encripta una cadena en un mensaje que también puede ser desencriptado con la misma clase.
Funciones:

- **bool add_new_key(std::string new_key)**
  Adiciona una nueva llave para poder encriptar las cadenas, se valida la llave de acuerdo al diccionario actual; se pueden adherir más llaves generando así una nueva llave.
- **void clear_keys()**
  Elimina todas las llaves que tiene actualmente el encriptador.
- **void set_dictionary(std::string new_dictionary)**
  Cambia el diccionario actual, esto eliminara la llaves que tiene actualmente el encriptador.
- **std::string get_dictionary()**
  Retorna el diccionario que esta usando el encriptador actualmente.
- **std::string encrypt(std::string message)**
  Codifica el mensaje y retorna el mensaje encriptado.
- **std::string decrypt(std::string encrypted_message)**
  Decodifica un mensaje encriptado y retorna el mensaje real.

Destructores:

- **~ Encryptor()**
  Elimina las llaves actuales y libera la memoria utilizada por el encriptador.

7.3 Lector de Archivos (LexRisLogic/FileStream.h)

7.3.1 Clase: FileStream

Clase que proporciona un lector de archivos texto plano, y funciones para controlar cada línea del archivo.

Funciones:

- **void set_path(std::string new_path)**
  Se asigna la ruta del archivo que sera creado o se leerá.
- **std::string get_path()**
  Retorna la ruta del archivo.
- **bool load()**
  Carga todas las lineas del archivo, que se encuentran en la ruta dada, en la estructura para que pueda ser manejada con las funciones de control línea a línea, retorna falso si no existe el archivo requerido; si se necesita borrar los cambios y recargar el archivo se puede llamar otra vez esta función.
- **bool save()**
  Guarda el archivo modificado, puede cambiarse la ruta del archivo antes de guardar, para evitar sobrescribir el archivo original.
- **void clear_file()**
  Limpia el archivo eliminando todas las líneas del archivo.
- **bool insert_line(unsigned int insert_position, unsigned int total_of_new_lines)**
  Inserta un número de líneas antes de la línea en la posición pasada por parámetro, retorna verdadero en caso de éxito.
- **bool remove_line(unsigned int remove_position)**
  Remueve la línea que se encuentra en la posición dada, retorna verdadero en caso de éxito.
- **unsigned int size()**
  Retorna el número de líneas que tiene actualmente el archivo.

Operadores:

- **std::string& operator [] (unsigned int line_position)**
  Retorna la línea que se encuentra en la posición dada, retorna por referencia para que puedan ser manipuladas las líneas del archivo.

Destructores:

- **~ FileStream()**
  Libera la memoria utilizada por el lector de archivos.

7.4 Matemática (LexRisLogic/Math.h)

7.4.1 Constante: const double MATH_PI

Valor de PI.

7.4.2 Función: Módulo (Matemática Discreta).

int mod(int dividend,int divisor)
Retorna el valor de $a \ mod \ b$, es decir, el valor que debería tomar a en un rango discreto de 0 a $b-1$.

7.4.3 Función: Módulo (Matemática Continua).

double range_mod(double dividend,double divisor)
Se mantiene la idea del módulo para ser usado en rangos discretos pero ahora con rangos continuos, por ejemplo 3.4 *mod* 3.3 es igual a 0.1, hacer trabajar a los números en un rango continuo de $0 \leq X < 3.3$, donde X sera el valor que tomaría a en base a b.

7.4.4 Función: Conversión de Grados Sexagesimales a Radianes

float sexagesimal_to_radian(float sexagesimal)
Convierte los grados sexagesimales enviados a radianes.

7.4.5 Función: Conversión de Radianes a Grados Sexagesimales

float radian_to_sexagesimal(float radian)
Convierte los radianes enviados a grados sexagesimales.

7.4.6 Función: Generadora de Semillas

void random_generate_new_seed()
Genera una semilla nueva para el generador de números aleatorios.

7.4.7 Función: Generar un Número Aleatorio

int random(int min_value,int max_value,bool include_max_value=false)
Retorna un número aleatorio que se encuentre en el rango de [min_value : max_value>, con el valor de include_max_value se puede incluir la posibilidad de que también se genere el número max_value.

7.4.8 Función: Colisión entre Segmentos

bool segment_collision(float ini_segment_1, float fin_segment_1,
float ini_segment_2,float fin_segment_2)
Usado para conocer si dos segmentos presentan colisión.

7.4.9 Función: Máximo Entero de un Número

int max_integer(float number)
Retorna el máximo entero del número flotante.

7.5 Separador de Cadenas (LexRisLogic/StringSplitter.h)

7.5.1 Clase: StringSplitter

Clase que representa a una función de separación de cadenas mediante un carácter.
Funciones:

- **void set_string(std::string new_string)**
  Cambia la cadena que será separada por la clase.
- **std::string get_string()**
  Retorna la cadena que será o ya ha sido separada por la clase.
- **bool split(char character='\n')**
  Separa la cadena de acuerdo al carácter enviado, por defecto se separa por saltos de líneas; todas estas subcadenas son guardadas en la estructura interna de la clase, retorna verdadero en caso de éxito.
- **unsigned int size()**
  Retorna el número de elementos que generó la separación de la cadena.
- **void clear()**
  Elimina todos los elementos que tiene la estructura interna de la clase.

Operadores:

- **const std::string operator [] (unsigned int index)**
  Retorna la i-ésima separación hecha por el separador sobre la cadena asignada.

Destructores:

- **~ StringSplitter()**
  Libera la memoria utilizada por la estructura interna que guarda todas las subcadenas generadas al separar la cadena.

7.6 Tiempo (LexRisLogic/Time.h)

7.6.1 Clase: Chronometer

Clase que representa al uso de un cronómetro para conocer el tiempo en segundos que pasa al realizar alguna actividad.
Funciones:

- **bool play()**
  Inicia el Cronómetro.
- **bool pause()**
  Pausa el Cronómetro.
- **bool stop()**
  Detiene el Cronómetro.
- **const double get_time()**
  Obtiene el tiempo acumulado por el cronómetro.
- **void clear()**
  Reinicia el cronómetro limpiando los datos guardados, puede ser usado en cualquier estado del cronómetro.

8 Notas

Toda esta documentación es dada para que se pueda conocer sobre el funcionamiento que brindan las Cabeceras de LexRis Logic, dirigida para la Versión v2.00.

9 License